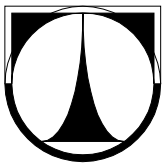
TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií



DIPLOMOVÁ PRÁCE

Liberec 2013 **Pavel Ozogán**

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics and Interdisciplinary Studies

Simulace Turingových strojů

The Turing machines simulation

STUDIJNÍ PROGRAM N2612 – ELEKTROTECHNIKA A INFORMATIKA

STUDY PROGRAMME N2612 – ELECTROTECHNOLOGY AND INFORMATICS

STUDIJNÍ OBOR 1802T007 – INFORMAČNÍ TECHNOLOGIE

STUDY BRANCH 1802T007 – INFORMATION TECHNOLOGY

DIPLOMOVÁ PRÁCE | DIPLOMA THESIS

Autor práce | Author — Pavel Ozogán

Vedoucí práce | Thesis supervisor — Ing. Lenka Kosková

LIBEREC 2013

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Pavel Ozogán**
Osobní číslo:	**M11000246**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Informační technologie**
Název tématu:	**Simulace Turingových strojů**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Proveďte rešerši existujících řešení.
2. Navrhněte a popište architekturu aplikace.
3. Nalezněte vhodné knihovny pro implementaci uživatelského rozhraní.
4. Navrhněte rozhraní pro zadání stroje a realizujte jej - aplikace by měla podporovat grafické zadání stroje a zadání formou binárního popisu.
5. Implementujte převod grafického zadání do binárního kódu a naopak.
6. Realizujte simulaci chování stroje; aplikace musí krokovat nebo animovat průběh zpracování slova strojem, kroky výpočtu budou přehledně zobrazovány.
7. Implementujte makra a ukládání, tak, aby do vyvíjených TS bylo možné vkládat makra již uložená v aplikace. Součástí aplikace budou nejméně tři základní makra pro TS.
9. Připravte vzorové řešení pro dva ukázkové stroje.
10. Doplňte nápovědu.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **60 stran**

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] Hopcroft, Motwani, Ullman: Automata theory, Languages and Computation; Addison Wesley; 2006; ISBN 0321455363.
[2] Jančar P.: Teoretická informatika (učební text); Ediční středisko VŠB - TUO; 2007; ISBN 978-80-248-1487-2.
[3] Chytil M.: Automaty a gramatiky; SNTL Praha, 1984.

Vedoucí diplomové práce: **Ing. Lenka Kosková**
Ústav nových technologií a aplikované informatiky

Datum zadání diplomové práce: **18. října 2012**

Termín odevzdání diplomové práce: **17. května 2013**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 18. října 2012

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Datum:

Podpis:

Abstrakt

Cílem práce je vytvořit interaktivní webovou aplikaci určenou pro simulaci Turingových strojů. V první části je popis Turingova stroje a porovnání existujících simulátorů. V další části je popis, jak v aplikaci Turingův stroj vytvořit a simulovat. Následuje je popis použité architektury aplikace. Poslední část tvoří srovnání vytvořeného simulátoru s existujícími.

Klíčová slova

turingův stroj, simulace, Javascript, MVC, návrhový vzor Observer

Abstract

Purpose of this diploma theses is to make interactive web application designed to simulate Turing machines. In first part is description of Turing machine and comparison of existing simulators. Next part describes how to create and simulate Turing machine in created application. User can draw machine in graphical form or as list of instructions. After that is description of used application architecture. In last part is compared created simulator with existings.

Keywords

turing machine, simulation, Javasript, MVC, Observer pattern

Obsah

Seznam zkratek

CSS	Cascading Style Sheets
GPLv2	GNU General Public License v2.0
HTML	HyperText Markup Language
MIT	Massachusetts Institute of Technology
MVC	Model–view–controller
PHP	PHP: Hypertext Preprocessor
SVG	Scalable Vector Graphics
TS	Turingův stroj
UI	User interface
VML	Vector Markup Language
XML	Extensible Markup Language

Seznam obrázků

1. Úvod

Turingův stroj je teoretický model počítače vymyšlený Alanem Turingem v roce 1936 na samém počátku počítačových věd. Tento pojem se používá jako synonymum pro algoritmus.

Na vysokých školách se o něm dodnes učí, protože je jedním ze základů teoretické informatiky a žádný prakticky sestavitelný stroj, který by měl větší výpočetní výkon, ještě nebyl vynalezen.

Po studentech je požadováno nějaký Turingův stroj vymyslet a nakreslit. Na papíře není simulace Turingova stroje moc efektivní a velmi jednoduše lze udělat chybu.

Z tohoto důvodu vznikla tato práce a jejím cílem je poskytnout studentům nástroj pro snadnější návrh Turingových strojů v grafickém tvaru a jejich simulaci. Nějaké simulátory už existují, ovšem žádný z nich není plně online bez instalace nějakých doplňků.

V úvodní části práce je definice samotného Turingova stroje, následuje rešerše a ohodnocení existujících simulátorů. Následuje popis funkcí aplikace a její ovládání. Dále je popsána architektura aplikace nástroje použité k napsání aplikace. V závěru je srovnání vytvořené aplikace s existujícími řešeními simulátoru Turingova stroje.

2. Turingův stroj

Kapitola předkládá formální definici Turingova stroje a popis jeho činnosti. Dále jsou představeny výsledky rešerše existujících simulátorů a jejich ohodnocení a srovnání s ideálním simulátorem.

2.1 Popis

Následující definice je převzata a přeložena z knihy [1].

Definice 1. *Turingův stroj je každá uspořádaná sedmice* $T = \{Q, \Sigma, P, \delta, q_0, B, F\}$, *kde:*

Q *je konečná množina stavů.*

Σ *je konečná množina vstupních symbolů.*

P *je kompletní množina možných symbolů na pásce.* Σ *je vždy podmnožinou* P.

δ *je přechodová funkce (zobrazení). Argumentem je dvojice* $\delta(q, X)$, *kde* q *je stav a* X *je symbol na pásce. Hodnota funkce pro* $\delta(q, X)$, *pokud je definována, je trojice* (p, Y, D), *kde*

p je nový stav z Q,

Y je zapisovaný symbol, který nahradí aktuální symbol na pásce a

D je směr pohybu hlavy (L/R).

$q_0 \in Q$ *je počáteční stav.*

B *je prázdný symbol, který je v* P, *ale ne v* Σ.

$F \subseteq Q$ *je konečná množina koncových stavů.*

Turingův stroj je teoretický model počítače. Skládá se z nekonečné pásky, hlavy a programu. Páska je rozdělena na buňky a každá buňka obsahuje jeden symbol.

Hlava je vždy nad jednou (aktuální) buňkou pásky, dokáže číst a zapsat symbol na aktuální buňku a posunout se o buňku doleva nebo doprava. Program pomocí přechodové funkce ovládá hlavu.

2.1.1 Postup výpočtu

Na počátku je na pásku zapsáno vstupní slovo, z obou stran řetězce jsou donekonečna prázdné symboly. Hlava je na prvním symbolu slova a aktuální stav programu je počáteční stav Turingova stroje.

V každém kroku výpočtu je přečten symbol z pásky, na základě kterého se rozhodne o další akci, kterou tvoří zapisovaný symbol, posun hlavy a nový stav.

Při přechodu do některého z koncových stavů je slovo strojem přijato. Pokud pro aktuální kombinaci stavu a symbolu na pásce neexistuje akce (p, Y, D), stroj slovo zamítá. Další možností je, že stroj bude cyklovat, což znamená, že se nikdy nezastaví.

2.1.2 Zápis programu Turingova stroje

Program je možno zapsat pomocí množin a přechodové funkce tak, jak jsou uvedeny v předchozí definici.

Další možností je grafický zápis formou grafu, kde každému stavu odpovídá vrchol grafu a přechody mezi stavy jsou znázorněny jako ohodnocené orientované hrany mezi uzly grafu. Stavy jsou obvykle očíslovány a u hran je zapsána podmínka a akce, která se tímto přechodem provede. Dále je nutno označit počáteční a koncové stavy. Pro úplnost dle definice je dobré uvádět množiny Σ a P.

Dle knihy [1] lze program Turingova stroje také zapsat jako binární řetězec. Stroje tím lze očíslovat, což se využívá v další teorii. Každý přechod ve tvaru $\delta(q_i, X_j) \rightarrow (q_k, X_l, D_m)$ je zakódován jako $0^i10^j10^k10^l10^m$. Jednotlivé přechody jsou pak odděleny jedničkami.

2.2 Makra

Při návrhu Turingova stroje se lze velmi často dostat do situace, kdy se nějaká skupina stavů častěji opakuje. Například při posunu hlavy na konec slova, nebo nějaký následující konkrétní znak.

Tyto opakující se skupiny stavů lze nahradit makrem. Pro samotný Turingův stroj se moc nemění, při přechodu do makra přejde na jeho počáteční stav a dále pokračuje jako obvykle. Po příchodu na koncový stav makra se vrátí do nadřazeného programu a pokračuje obvyklým způsobem dále.

Makra lze i vnořovat do sebe, ovšem nelze použít rekurzi. Ta by znamenala nekonečný počet stavů, což by odporovalo definici, že Q je konečná množina stavů.

I při vytváření nejjednodušších maker jako například posun na následující symbol „A“ nebo na konec slova je vhodné mít znaky zastupující více symbolů. Pak nebude potřeba psát například přechod pro každý symbol, který není „A“. Navíc by tak toto makro bylo pro každý stroj jiné, záviselo by na použité abecedě P.

2.3 Ideální simulátor

Pro srovnání jednotlivých simulátorů jsem vytvořil seznam požadavků pro ideální simulátor, které jsou ohodnoceny podle důležitosti. Ideální simulátor bude ohodnocen maximem, což je sto bodů.

- Simulace Turingova stroje po krocích (max. 25 bodů)
- Webová aplikace bez pluginů (20 bodů) / Flash (18 bodů) / Java-applet (15 bodů)
- Grafická podoba stroje (max. 20 bodů)
- Uložení a načtení do souboru (15 bodů) / Uložení a načtení přes schránku (10 bodů)
- Jednoduché ovládání (max. 10 bodů)
- Makra (5 bodů)
- Několik ukázkových příkladů (1 bod za každý, max. 5 bodů)

2.4 Srovnání některých existujících simulátorů

Ideální simulátor by měl být volně dostupný na webu z jakékoliv platformy a jako jazyk by se pro něj nejlépe hodil JavaScript, který by umožňoval i grafické rozhraní. Jako další možnosti připadají v úvahu Flash a Java-applety, které je ovšem nutno doinstalovat.

Dále následuje stručný popis nalezených simulátorů s jejich bodovým ohodnocením v závorkách. Srovnány jsou dle získaných bodů.

Turing in a Flash Webový simulátor s použitím technologie Flash (18), jako jediný online umožňuje jednoduchý intuitivní grafický návrh, ale má několik nedostatků (15). Přechody ze stavu mohou obsahovat pouze příkaz k zapsání znaku anebo posunutí hlavy, obvykle jsou ale potřeba obě činnosti a je nutno pro každý takový přechod mít stav navíc. Umožňuje krokování výpočtu, ale po resetování stroje je nutné znovu zapsat vstupní slovo na pásku a označit počáteční stav (20). Umožňuje načíst/uložit stroj nebo pásku jako soubor v XML formátu (15). Ovládání je intuitivní (10), i když není k dispozici žádný vytvořený stroj. (Celkem 78 bodů.)

Tm - The Turing Machine Simulator Desktopový simulátor napsaný v Javě, který umožňuje grafický návrh (20). Pro návod k vytvoření přechodu ze stavu do stavu je ale nutno si přečíst nápovědu, jinak ovládání není složité (8). Po nahrání přiložených příkladů (5) se hlava nastavuje do pozice za posledním symbolem vstupního slova, někdy ale také na první nebo druhý. Umožňuje simulaci po krocích (25), makra (5) i ukládání do souborů (10). (Celkem 78 bodů.)

Turing machine simulator [JavaScript] Asi nejlepší webový simulátor v JavaScriptu (20) s jednoduchým ovládáním (10) a zadáváním stroje v textovém tvaru. Umožňuje krokování a pozastavení simulace (25). Uložení a načtení stroje lze pouze přes schránku (10). K vyzkoušení je v programu celkem 7 příkladů (5). (Celkem 70 bodů.)

turingsimulatorv5 Další JavaScriptový (20) simulátor s jednoduchým ovládáním (10) a textovým tvarem programu. Uložení a načtení pouze přes schránku (10). K dispozici 6 příkladů (5), simulaci lze krokovat (25). (Celkem 70 bodů.)

xTuringMachine Lab Java-applet (15) s intuitivním ovládáním (10). Zadávání programu je pomocí tabulky. Má ale omezení na maximálně 26 stavů a znaky pásky omezeny na „#$01xyz“. Nelze tedy vytvořit složitější stroje. Ze všech nalezených simulátorů má největší počet příkladů (5), pár jich lze nahrát přímo z nabídky v programu a ostatní lze stáhnout a nahrát přes soubory (15). Umožňuje simulaci po krocích, chybí ale tlačítko pro reset (22). (Celkem 67 bodů.)

Visual Turing Machine Simulátor v Javě, ve kterém se program zadává graficky v nestandardní podobě (15). Je tedy vhodné si přečíst nápovědu (5). Jednotlivé instrukce (zápis znaku nebo posun hlavu) se spojují pomocí šipek, u kterých může být uvedena podmínka (znak na pásce). Jako jeden z mála umožňuje používat makra (5). Umožňuje ukládání do souboru (15) a krokování simulace (25). (Celkem 65 bodů.)

Webový simulátor Turingova stroje Bakalářská práce o vytvoření simulátoru pomocí Java-appletu (15). Jako jediný simulátor částečně podporuje nedeterministické stroje, pokud je nalezeno více možností, tak se jedna z nich náhodně vybere. Správně by se ale měly vyzkoušet postupně všechny možnosti. Simulaci stroje je možné krokovat (25). Turingův stroj je nutno importovat z textového souboru, nelze jeho program napsat v aplikaci (12). Lze ale vybrat některý z předpřipravených (5). Ovládání je jednoduché, jen pro vytvoření vlastního stroje je nutno nastudovat nápovědu (5). (Celkem 62 bodů.)

TuringSim Další z webových simulátorů v JavaScriptu (20). Ovládání je jednoduché, jen je nutné si přečíst v nápovědě, jak stroj zapsat (5), protože nemá ani jeden příklad stroje. Program se zadává v textovém tvaru, uložení a načtení lze pouze přes schránku (10). Simulaci lze krokovat (25). (Celkem 60 bodů.)

Simulátor Turingových strojů v1.1 Jako jediný je napsán čistě v PHP, fungovat tedy bude v jakémkoliv webovém prohlížeči (20). Z toho také ovšem plyne, že stroj se zadává pouze tabulkou a postup výpočtu se také zobrazuje jako tabulka, a to všechny kroky najednou (15). Všechny stroje jsou uloženy na serveru a není možné jednoduchý import nebo export (5). Ovládání je jednoduché (10) a na serveru je několik vytvořených příkladů (5). (Celkem 55 bodů.)

Uber Turing Machine Placený program pro Windows určený spíše pro simulaci než návrh, nemá totiž možnost grafického návrhu, ale zobrazuje například statistiku využití jednotlivých přechodových pravidel. Možnost stažení trial verze na 30 dní. Umožňuje krokování (25) a ukládání do souborů (15). Ovládání je jednoduché (10) a k dispozici je několik příkladů strojů (5). (Celkem 55 bodů.)

Alan Turing Internet Scrapbook Simulátor napsaný v JavaScriptu (20) s jednoduchým ovládáním (10), který ale neumožňuje zadat vlastní stroj, pouze simuluje 3 předpřipravené (3). Program je zobrazován jako tabulka. Dle nápovědy by měl umožňovat krokování, ale tlačítko „step“ nereaguje (10). (Celkem 43 bodů.)

Turing Machine Simulator Java-applet (15), ve kterém ale v aktuální verzi Javy (7 update 17) nefunguje zobrazování pásky, takže program není moc použitelný pro simulaci (10). Zabudováno má v sobě 6 příkladů (5) a program se zadává pomocí textové tabulky, ukládat lze pouze přes schránku (10). Ovládání je jednoduché (10). (Celkem 35 bodů.)

2.4.1 Seznam odkazů na nalezené simulátory

- Turing in a Flash
  `http://turinginaflash.com/index.php`
- Tm - The Turing Machine Simulator
  `http://www.cs.utah.edu/~dhenders/cs4500/`

- Turing machine simulator [JavaScript]
  http://morphett.info/turing/turing.html
- turingsimulatorv5
  http://www.willamaze.eu/wp-content/uploads/2009/07/turingsimulatorv5.html
- xTuringMachine Lab
  http://math.hws.edu/TMCM/java/labs/xTuringMachineLab.html
- Visual Turing Machine
  http://sourceforge.net/projects/visualturing/
- Webový simulátor Turingova stroje
  http://is.muni.cz/th/72784/fi_b/
- TuringSim
  http://www.kishwaukeecollege.edu/faculty/dklick/cis119/TuringSim.html
- Simulátor Turingových strojů v1.1
  http://ui.glimpse.cz/sts/
- Uber Turing Machine
  http://www.superutils.com/products/uber-turing-machine/
- The Alan Turing Internet Scrapbook
  http://www.turing.org.uk/turing/scrapbook/tmjava.html
- Turing Machine Simulator [Java-applet]
  http://ironphoenix.org/tril/tm/

2.5 Výsledky rešerše

Existuje velké množství webových simulátorů, pouze jeden z nalezených ale má možnost grafického návrhu stroje. Obvykle je mají pouze desktopové programy, které se musí nainstalovat a dva z výše uvedených ještě potřebují nainstalovat Javu.

Tab. 2.1: Bodové ohodnocení simulátorů, K - krokování, W - webová aplikace, G - grafický návrh, U - uložení a načtení ze souboru, J - jednoduché ovládání, M - makra, P - několik příkladů

Jméno simulátoru	K	W	G	U	J	M	P	Celkem
Ideální simulátor	25	20	20	15	10	5	5	100
Turing in a Flash	20	18	15	15	10	0	0	78
Tm - The Turing Machine Simulator	25	0	20	15	8	5	5	78
Turing machine simulator [JavaScript]	25	20	0	10	10	0	5	70
turingsimulatorv5	25	20	0	10	10	0	5	70
xTuringMachine Lab	22	15	0	15	10	0	5	67
Visual Turing Machine	25	0	15	15	5	5	0	65
Webový simulátor Turingova stroje	25	15	0	12	5	0	5	62
TuringSim	25	20	0	10	5	0	0	60
Simulátor Turingových strojů v1.1	15	20	0	5	10	0	5	55
Uber Turing Machine	25	0	0	15	10	0	5	55
The Alan Turing Internet Scrapbook	10	20	0	0	10	0	3	43
Turing Machine Simulator [Java-applet]	10	0	0	10	10	0	5	35

3. Ovládání a možnosti simulátoru

V této kapitole je ukázáno vlastní řešení simulátoru Turingova stroje. Při návrhu grafického rozhraní jsem se inspiroval tradičním grafickým zápisem programu Turingova stroje na papír. Rozvržení prvků aplikace se podobá jiným prostředím pro psaní programů (ovládání simulace pomocí tlačítek nahoře, výstup dole, vlevo editace vlastností).

Stroje je možné zadávat i v textové (tabulkové) a binární podobě. Textový formát jsem vytvořil tak, aby nebylo složité požít stroj z jiného existujícího simulátoru, který program ukládá v podobném tvaru. Souřadnice jednotlivých prvků se pak automaticky dopočítají. Binární tvar slouží hlavně pro ukázku, že Turingovy stroje lze očíslovat. Případně je možné využít převodu na binární tvar a zpět pro odstranění maker a znaků `#`.

Pro usnadnění práce je možno použít opakující skupiny stavů najednou jako makra. Při dodržení podmínky, že se hlava pohybuje vždy doprava, ho lze jednoduše použít i jako simulátor deterministických konečných automatů.

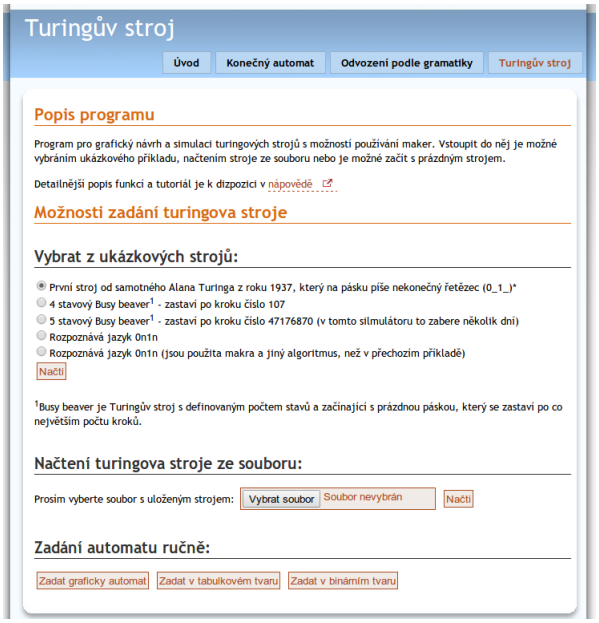
V kapitole následuje popis obrazovek simulátoru, dále je na příkladu ukázán postup vytvoření jednoduchého Turingova stroje, jeho otestování pomocí simulace a použití tohoto Turingova stroje jako makro v jiném stroji. V poslední části je podrobnější popis jednotlivých funkcí programu včetně popisu formátu souborů pro ukládání Turingových strojů.

Ve čtvrté kapitole je pak popsáno, jak byla aplikace naprogramována. Jsou uvedeny použité technologie a knihovny a popsány použité návrhové vzory.

Simulátor je součástí serveru `kaja.nti.tul.cz`, na kterém je několik pomůcek pro výuku. Na stránku simulátoru se lze dostat odkazem z hlavní stránky serveru, nebo přímo pomocí adresy `kaja.nti.tul.cz/~ozogan/turing/`.

3.1 Popis obrazovek programu

3.1.1 Úvodní stránka



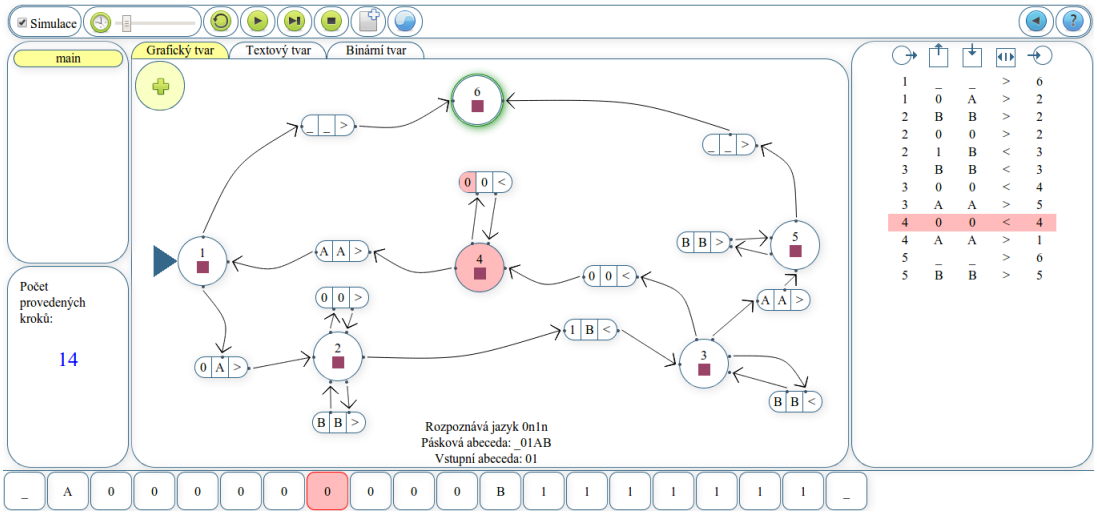
Obr. 3.1: Úvodní stránka dává na výběr, jaký Turingův stroj v aplikaci načíst.

Na úvodní stránce simulátoru je na výběr několik možností, s jakým Turingovým strojem program spustit. Je možno využít 5 předem vytvořených a funkčních Turingových strojů. Další možností je nahrát Turingův stroj ze souboru a poslední je začít navrhovat nový stroj, který už bude mít vytvořený počáteční a koncový stav. Úvodní stránka je ukázána na obrázku 3.1.

3.1.2 Hlavní stránka

Po přechodu na hlavní stránku je možné načtený Turingův stroj upravovat a simulovat, jak je popsáno v dalších částech. Hlavní stránku lze vidět na obrázku 3.2.

V horní části stránky jsou převážně prvky pro ovládání simulace, nejvíce vlevo je to její zapnutí, dále potom směrem vpravo je to nastavení rychlosti ( ), tlačítka pro reset ( ), spuštění ( ), krokování ( ) a zastavení ( ). Dalším je tlačítko pro přidání/nahrání makra z aplikace ( ). Poslední v levé části je tlačítko ( )



Obr. 3.2: Hlavní stránka simulátoru. V horní části je ovládání simulátoru, vlevo je seznam maker, vpravo seznam instrukcí aktuálního programu, uprostřed je graficky zobrazený program a ve spodní části je páska

pro automatické přerovnání grafické podoby programu Turingova stroje tak, aby se vešel na obrazovku a nebylo v něm nejlépe žádné křížení šipek. Jako poslední jsou pak nahoře vpravo tlačítko pro návrat na úvodní stránku ( ) a k nápovědě ( ).

V levém sloupci je seznam, ve kterém je vždy hlavní program Turingova stroje a případně makra, která tento stroj může použít. Vybraný program nebo makro je zobrazen v prostřední části obrazovky. V dolní části levého sloupce jsou vždy informace o zvoleném prvku a nějaké možnosti jeho editace, jako třeba změna jména stavu, nebo makra.

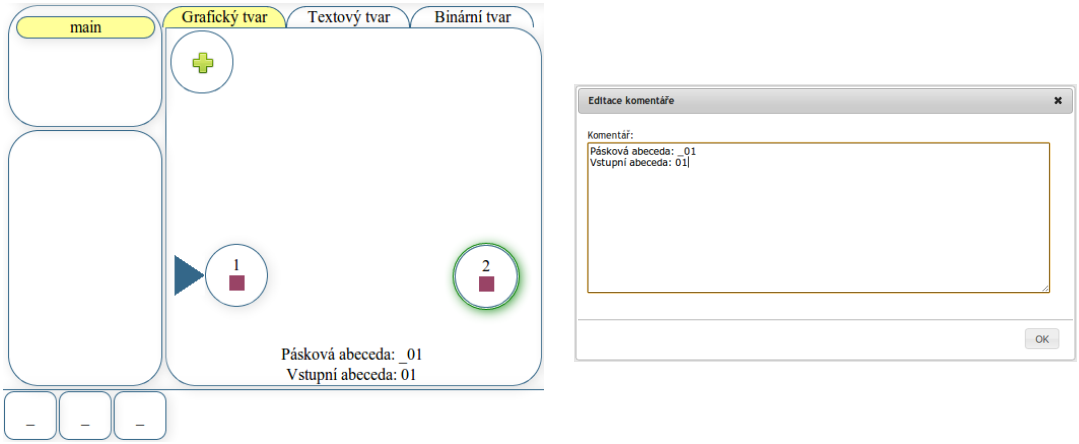
V pravém sloupci jsou zobrazeny všechny instrukce zobrazovaného programu nebo makra v přehledné podobě pomocí tabulky. Jednotlivé sloupce jsou označeny pomocí symbolů pro výchozí stav ( ), podmínku ( ), zapisovaný znak ( ), směr pohybu hlavy ( ) a nový stav ( ). Při simulaci se vždy zvýrazní následující prováděná instrukce.

Ve spodní části je zobrazena oboustranně nekonečná páska, prázdná políčka jsou zobrazena se znakem _. Vstupní slovo je vždy ohraničeno těmito znaky zleva a zprava, které pokračují donekonečna, ale z úspory místa jsou vždy zobrazeny pouze první vlevo a vpravo.

Program Turingova stroje je zobrazován v prostřední části obrazovky, záložkami vlevo nahoře lze přepínat mezi grafickým, textovým a binárním zobrazením. Ukázky zobrazení jednotlivých částí programu jsou na obrázcích 3.9 až 3.14. Příklad kompletního Turingova stroje je v části 3.2.1.

3.2 Návrh Turingova stroje

3.2.1 Příklad vytvoření Turingova stroje s makry a jeho simulace

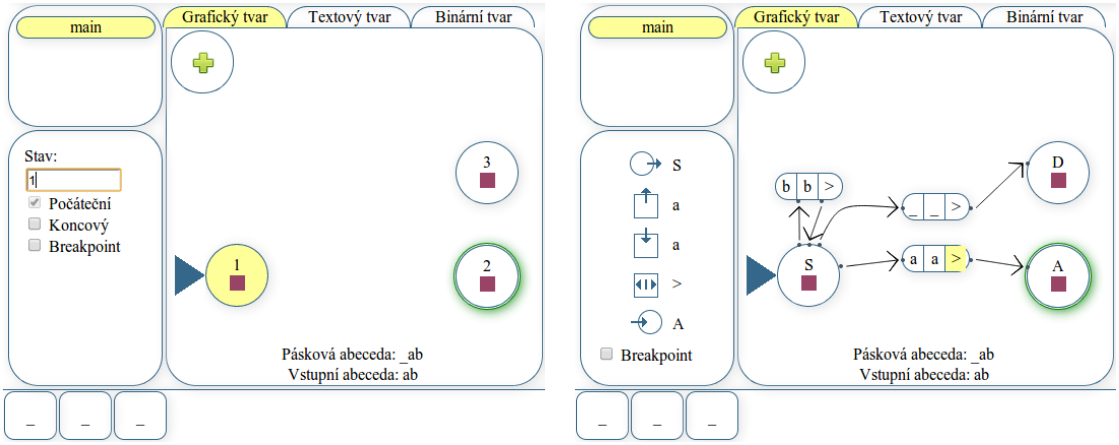


(a) (b)

Obr. 3.3: Vlevo (a) je nově vytvořený Turingův stroj s počátečním a koncovým stavem, vpravo (b) je dialog pro změnu komentáře a abeced.

Zadání: Sestrojte Turingův stroj akceptující slova obsahující minimálně jeden znak a, pracující nad abecedou $\Sigma = \{a, b\}$. Jako druhý sestavte stroj pracující nad abecedou $\Sigma = \{a, b, c\}$ akceptující slova s minimálně třemi znaky b.

Vytvoření nového stroje: Z úvodní stránky aplikace si lze vybrat, s jakým automatem začít. V tomto příkladu to je dole ruční zadání nového stroje v grafické podobě. Načte se hlavní stránka programu se strojem, který tvoří nespojený počáteční a koncový stav.



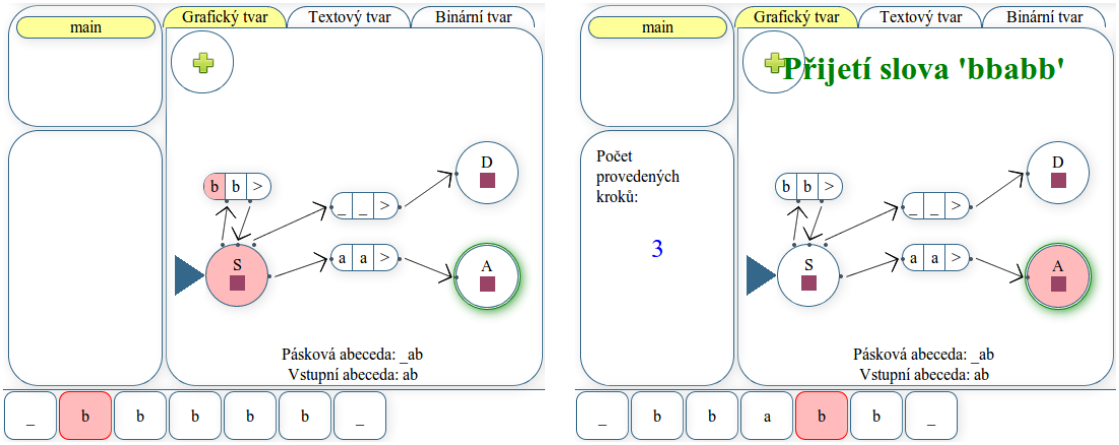
(a) (b)

Obr. 3.4: Vlevo (a) byl přidán stav číslo 3, stav číslo 1 je označený myší a vlevo ja možné změnit jeho jméno. Vpravo (b) byly přejmenovány stavy a doplněny přechody, tím je program tohoto Turingova stroje dokončen.

Aktuální stav je na obrázku 3.3(a). V komentáři jsou vidět použité abecedy se znaky 0, 1, které se ale neshodují se zadáním. Aby bylo možné znaky ze zadání zapsat do stroje a na pásku, tak je nutné tyto abecedy změnit. Udělá se to pomocí dvojkliku na komentář, po kterém se zobrazí dialogové okno s nabídkou změny komentáře. Zobrazeno je na obrázku 3.3(b). Součástí komentáře jsou i abecedy stroje, je tedy nutno zaměnit znaky `01` za `ab` a kliknout na OK.

Nový stav se přidá přetáhnutím nového stavu vlevo nahoře ( ) do pozice, kde má být přidán. Nové stavy se automaticky číslují od jedničky nahoru, stav ale lze pojmenovat vlastním jménem. Po kliknutí na stav je možné v levé části okna, jak je ukázáno na obrázku 3.4(a), přepsat jeho jméno na nové. V tomhle tutoriálu je přejmenován stav `1` na `S`, stav `2` na `A` a nový stav `3` na `D`. Není ale třeba je přejmenovávat, na funkci stroje se tím nic nemění.

Přechody ze stavu do stavu se přidávají pomocí myši přesunutím ze čtverečku pod názvem stavu do cílového stavu. Ze stavu `S` je potřeba udělat přechod do stavu `A` a z `S` do `D`. Podobně se vytvoří i přechod z `S` do `S`. Každý stav a přechod lze myší jednoduše přesunovat, je tedy možné posledně přidaný přechod posunout nahoru, aby se nepřekrýval se stavem `S`.



(a) (b)

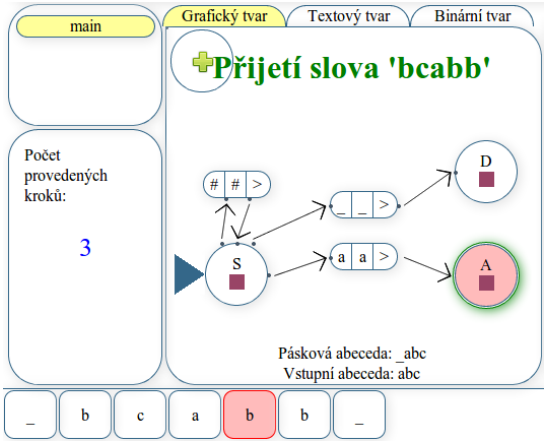
Obr. 3.5: Vlevo (a) je situace po zapnutí simulace. Je zvýrazněn počáteční stav a první znak slova. Vpravo (b) je situace po přijetí slova „bbabb“.

V dalším kroku je potřeba jednotlivým přechodům nastavit podmínky a prováděné akce. To se provede kliknutím na příslušný prvek a stisknutím klávesy pro požadovaný znak, nebo šipku pro směr vlevo/vpravo. Výsledný stroj je zobrazen na obrázku 3.4(b). Po doplnění znaků a šipek dle tohoto obrázku je Turingův stroj hotový a následuje jeho otestování a pak upravení na makro.

Simulace stroje: Jako první je třeba nějaký řetězec na pásku, to se provede kliknutím na prostřední políčko pásky a napsáním požadovaného slova na klávesnici. V tomto příkladu se jako první bude testovat slovo `bbbbb`.

Režim simulace se zapíná zaškrtnutím křížku vlevo nahoře. Graficky se zvýrazní počáteční stav, první znak slova a podmínka následujícího prováděného přechodu, jak je ukázáno na obrázku 3.5(a). Samotná simulace se spustí tlačítkem ( ) nahoře a v horní části okna se objeví zpráva, že slovo bylo zamítnuto. Stroj se tedy chová správně, protože ve slově není znak `a`. Aby pro jeho přidání je nutné vypnout simulaci, kliknout na požadované políčko a zapsat ho pomocí klávesnice stisknutím `a`. Po opětovném zapnutí a spuštění simulace je vidět, že slovo bylo přijato, jak je ukázáno na obrázku 3.5(b).

Pro danou abecedu stroj funguje, ale při rozšíření jeho abecedy o znak `c` a zapsání



Obr. 3.6: Znak „b“ byl nahrazen znakem „#“, stroj takto bude fungovat pro jakoukoliv větší abecedu a bude tak možné ho použít jako makro. Přijetí ukázáno na slově uvbcabb.

tohoto znaku do slova před `a`, stroj slovo zamítne, i když podle slovního popisu by ho měl přijmout. Jedna možnost by byla přidat podmínku pro `c` ze stavu `S`. Stále by ale stroj byl závislý na abecedě, proto jsem zavedl znak `#`, který může být použit místo tohoto `c` a všech dalších, které by původní abecedu mohly rozšířit.

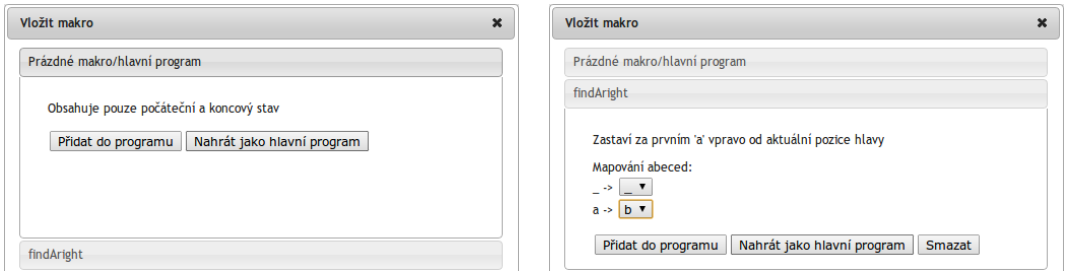
Ve stroji stačí v jediném přechodu se znaky `b` tyto znaky přepsat na `#`. Stroj pak bude přijímat slova, která obsahují minimálně jedno `a`, a nebude záviset na použité abecedě. Bude možné ho využít jako makro v jiném stroji. Výsledné makro s přijetím slova je zobrazeno na obrázku 3.6.

Do komentáře je také vhodné slovně zapsat, co vlastně stroj nebo makro dělá:

```
Zastaví za prvním 'a' vpravo od aktuální pozice hlavy
```

Uložení a použití stroje jako makra: Aby bylo toto makro použitelné v jiném stroji, je potřeba ho nejprve uložit. To se provede nejprve kliknutím na jeho jméno, což je prozatím `main`, vlevo v seznamu programů. O něco níže pak lze jeho jméno změnit podobně jako jméno stavu a poté lze stisknutím příslušného tlačítka uložit makro do aplikace. V tomto příkladu je vhodné toto makro přejmenovat například na `findAright` a uložit popsaným způsobem.

Teď je potřeba vytvořit prázdný hlavní program a do něj toto makro nahrát.



(a) (b)

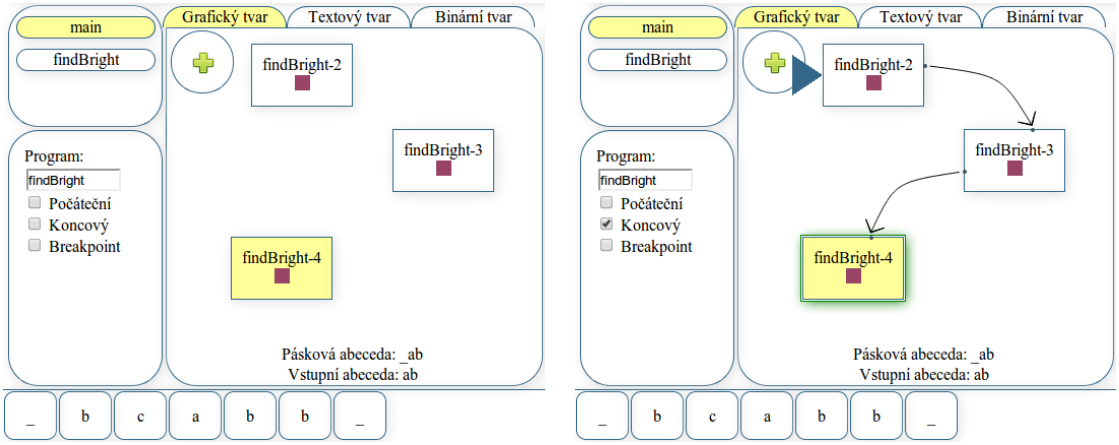
Obr. 3.7: Zobrazen je dialog vyvolaný příslušným tlačítkem v horní liště. Vlevo (a) je je vybrána volba pro přidání prázdného makra. Po kliknutí na makro v dolní části se dialog změní (b) a je možné přidat toto makro. Současně lze přemapovat abecedy makra na abecedy aktuálního stroje.

Kliknutím na tlačítko přidat ( ) se otevře dialogové okno jako na obrázku 3.7(a). Zde lze přidat prázdné makro, nebo nahrát nový prázdný hlavní program, nyní je správná druhá možnost. Následně je třeba změnit abecedy zpátky na `abc`, což se provede stejně jako minule.

Dalším krokem je přidání makra. Po kliknutí na tlačítko přidat se objeví nabídka jako minule a v místě pod tlačítky jsou jména maker, která lze přidat do programu. Po vybrání vytvořeného makra se rozbalí jeho detaily, jako na obrázku 3.7(b). Zde lze namapovat znaky makra na znaky hlavního programu. V tomto případě je potřeba se zastavit na znaku `b`, ale makro zastavuje na znaku `a`. Je tedy třeba tento znak zaměnit za `b`, což se provede záměnou znaku v rozbalovacím seznamu vpravo od `a`. Pak už stačí kliknout na tlačítko „Přidat do programu“.

Znaky v uvozovkách v komentáři a v přechodech stroje se patřičně změní, ale název zůstane stejný. Protože teď makro hledá `b`, je vhodné ho přejmenovat na `findBright`. Do programu mezi stavy se makro přidá jeho přetažením myší ze seznamu maker vlevo. Protože podle zadání má přijímané slovo obsahovat tři znaky `b`, makro tedy bude potřeba třikrát. Přebytečný koncový a počáteční stav lze smazat po jeho označení klávesou DELETE. Výsledek by měl vypadat podobně jako na obrázku 3.8(a).

Jednotlivé instance makra `findBright` jsou odlišeny číslem, za jeho názvem.



(a) (b)

Obr. 3.8: Vlevo (a) je stroj s třemi instancemi makra „findBright“, které vznikly přetažením z levého seznamu. Vpravo (b) jsou makra spojena a jedno je označeno jako počáteční, další jako koncové.

Makra lze stejně stavy označit jako počáteční nebo koncová. Jedno tedy bude počáteční a jedno koncové. Pak je nutné je propojit, jak je ukázáno na obrázku 3.8(b). Vytvoří se ale přechody s podmínkami. Po označení podmínky, nebo jiné části přechodu ho klávesou DELETE ho lze smazat a zůstane pouze šipka z jednoho makra, do jiného. Toto přímé spojení může existovat pouze mezi dvěma makry a vymaže se po dvojkliku na něj. Tím je stroj, podle druhého bodu zadání hotov. Na pásce by mělo být stále slovo `bcabb`, takže po zapnutí a spuštění ( ) simulace stroj toto slovo přijme, což je očekávaný výsledek.

3.2.2 Popis formátu souborů

Každý Turingův stroj nebo jeho makro lze uložit do dvou typů souborů. V textovém tvaru se zachovají všechna makra a pozice všech prvků na obrazovce v grafickém návrhu. Binární tvar odstraní všechna makra a souřadnice jednotlivých stavů a přechodů, funkčnost Turingova stroje ale zůstane zachována. Je tedy vhodnější Turingovy stroje ukládat v textovém tvaru.

Nahrát soubor lze tlačítkem nahrát na příslušné záložce, stejně tak uložit. Do programu se stroj uloží po změně textu a kliknutí myší mimo textové pole. Pokud

vstup neodpovídá pravidlům, objeví se seznam řádků s chybami.

Textový tvar

V textovém tvaru je program Turingova stroje zapsán jako seznam instrukcí (přechodů). Pro každý přechod a stav mohou být uvedeny souřadnice. Počáteční a koncové stavy programu jsou uvedeny na jeho začátku na řádku začínajícím #, za kterým následuje jméno příslušného programu nebo makra.

Následuje příklad Turingova stroje v textovém tvaru, který bude vypisovat slovo „(BA)*“ zprava doleva:

```
#Main aLeft-0
;Vypisuje '(BA)*' zprava doleva
;Pásková abeceda: _AB
;Vstupní abeceda:
aLeft-0 73 138
aLeft-1 478 139
aLeft-0 _ B < aLeft-1 301 194
aLeft-1 _ B < aLeft-0 301 145

#aLeft 1 2
;Zapsání 'A' a posun doleva
1 100 200
2 300 200
1 _ A < 2 223 245
```

Každý řádek začínající znakem `#` začíná nový program, jako první je hlavní program a následující jsou makra. Po znaku `#` ihned následuje název programu a poté jsou mezerami odděleny jeden počáteční uzel a libovolný počet koncový uzlů.

```
#jmeno_programu pocatecnistav [koncovystav]...
```

V příkladě je tedy `Main` hlavní program a `aLeft` jeho makro. V hlavním programu

je počáteční uzel makro `aLeft-1` a nemá koncový stav. V makru je počáteční uzel `1` a koncový `2`.

Řádky začínající znakem `;` jsou komentáře a jejich součástí je definice vstupní a páskové abecedy. V příkladě nahoře může být řádek se vstupní abecedou vynechán, protože vstupní abeceda je prázdná.

Mezi názvem programu a názvem následujícího programu následují řádky s popisem jednotlivých přechodů a souřadnic stavů. Při psaní těchto souborů ručně není nutné souřadnice u přechodů a celé řádky se stavy vypisovat.

Řádky s popisem uzlů jsou složeny z jeho jména a dvou souřadnic (`x` a `y`). Údaje jsou odděleny mezerami. Jméno konkrétní instance makra musí být utvořeno stejně, jako se to děje automaticky v grafickém návrhu, tedy jméno programu makra následované pomlčkou a číslem.

```
jmeno_stavu souradnice_x souradnice_y
jmeno_makra-cislo_jeho_instance souradnice_x souradnice_y
```

U přechodů je na řádku pět nebo sedm údajů také oddělených mezerami:

- uzel, ze kterého přechod vede
- podmínka přechodu
- zapisovaný znak
- posun doleva nebo doprava (lze použít i znaky 'l/r')
- uzel, na který přechod vede
- nepovinné souřadnice „x“ a „y“

```
vychozi_uzel podminka zapis posun cilovy_uzel [sour_x] [sour_y]
```

Binární tvar

V binárním tvaru je program Turingova stroje tvořen řetězcem jedniček a nul. Předchozí příklad by v binárním tvaru vypadal následovně (pro větší přehlednost zapsáno s mezerami):

```
111 01010010100 11 00101000101000 11 000101001010000
11 00001010001010 11 000001010100100000000
11 000000101010010000000 111
```

Řetězce `111` značí začátek a konec programu. Dvojice jedniček `11` odděluje jednotlivé přechody a každý přechod ve tvaru $\delta(q_i, X_j) \rightarrow (q_k, X_l, D_m)$ je zakódován jako $0^i10^j10^k10^l10^m$.

Jako počáteční stav je určen ten s nejmenším číslem a jako koncový ten s největším. Pokud by nějaký Turingův stroj neměl koncový stav tak je nutno ho přidat spolu s nedosažitelným stavem, který na koncový stav vede libovolným přechodem.

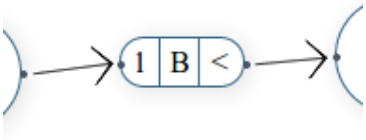
3.3 Grafický návrh Turingova stroje

Při vytváření nového programu pro Turingův stroj se načte základní šablona, která obsahuje počáteční a koncový stav, má prázdnou vstupní abecedu a páskovou abecedou tvoří pouze znak pro prázdný symbol _. Nejprve je tedy nutno tyto abecedy rozšířit, aby bylo možné na pásku a do programu zapisovat další symboly. Provede se dvojklikem na komentář a jeho úpravou.

Nové stavy lze přidávat ze „zásobníku“ vlevo nahoře přetažením myší do požadovaného místa. Označením stavu se o něm vlevo dole objeví informace a je tam i možnost je měnit. Lze měnit jméno stavu, při pokusu o zapsání jména, které už existuje, zůstane nastaveno poslední, které duplicitní nebylo. Protože každý Turingův stroj musí mít přesně jeden počáteční stav, tak tuto položku u aktuálního počátečního stavu nelze odškrtnout. Zmizí pouze nastavením jiného stavu jako počátečního nebo jeho smazáním. Dále je možné u stavu nastavit, jestli bude koncový nebo breakpoint. Zobrazení jednotlivých typů stavů je na obrázku 3.9.

Obr. 3.9: Nejvíce vlevo je obyčejný stav, vpravo od něj je počáteční stav, následující je koncový stav a jako poslední je zobrazen stav s breakpointem. Pomocí čtverečku pod názvem lze myší vytvářet přechody do dalších stavů nebo maker.

Nový přechod se vytvoří myší přetáhnutím šipky ze čtverečku ve zdrojovém stavu do cílového stavu. Nový přechod má vždy jako podmínku a zápis prázdný symbol a posun doprava. Úprava přechodové podmínky a akce se provede kliknutím na daný prvek a zapsáním znak pomocí klávesnice, u směru to jsou šipky vpravo/vlevo. Pokud znak není v páskové abecedě, tak zapsat nepůjde a zobrazí se upozornění o chybě. V levé části obrazovky lze aktuálně označený přechod nastavit jako breakpoint. Stroj se pak zastaví po vykonání akce na tomto přechodu. Ukázka zobrazení přechodu je na obrázku 3.10.

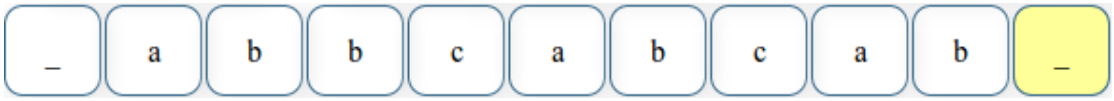


Obr. 3.10: Zobrazen je přechod z jednoho stavu do druhého. V tomto případě se provede, pokud bude na pásce znak „1“ a jako aktuální bude stav vlevo. Hlava na pásku zapíše „B“ a posune se o políčko doleva, stav vpravo se nastaví jako aktuální.

Přechody i stavy lze také tažením myší přesunovat. Při posunu stavy se vždy o polovinu vzdálenosti posouvají i k němu připojené přechody.

U pásky se na ní myší umístí kurzor a zapisování funguje podobně při psaní obyčejného textu, klávesy DELETE a BACKSPACE fungují očekávaným způsobem. Páska je nekonečná z obou stran, znaky _ vlevo a vpravo od slova reprezentují prázdné znaky od konce slova až do nekonečna. Ukázka zobrazení pásky je na obrázku 3.11. Na něm je kurzor na znaku za slovem, při stisku klávesy s písmenem

se tento znak zapíše na tuto pozici a kurzor se přesune o znak doprava na nově vytvořený znak _.



Obr. 3.11: Zobrazeno je slovo uvabbcabcab na pásce. Vlevo a vpravo od slova jsou znaky „_“, reprezentující prázdné symboly odpovídajícím směrem až do nekonečna. Kurzor je za posledním znakem.

Mazání se provádí označením daného prvku myší a stiskem klávesy DELETE, spolu se smazáním stavu nebo makra se smažou i přechody, které do něj, nebo z něj, vedou.

3.3.1 Přidávání maker

Po prvním načtení programu do prohlížeče se vytvoří několik užitečných maker (včetně prázdného), které lze jednoduše přidat do vytvářeného Turingova stoje.

Po kliknutí na ikonu pro přidání makra se zobrazí seznam se všemi makry a programy uloženými v aplikaci, která zůstávají uchovaná i po vypnutí prohlížeče a počítače. Makro lze přidat do stávajícího stroje, nebo ho použít jako nový samostatný stroj, kliknutím na příslušné tlačítko. Z tohoto seznamu je lze i mazat.

Před přidáním makra lze nastavit, jak se bude jeho abeceda mapovat na páskovou abecedu stroje. U makra, které se zastaví nad prvním znakem `b` vpravo od hlavy tak lze změnit jeho symbol ’b’ na nějaký, který je potřeba a současně je v páskové abecedě vytvářeného stroje. Nemusí se tedy vytvářet makro pro každý symbol zvlášť.

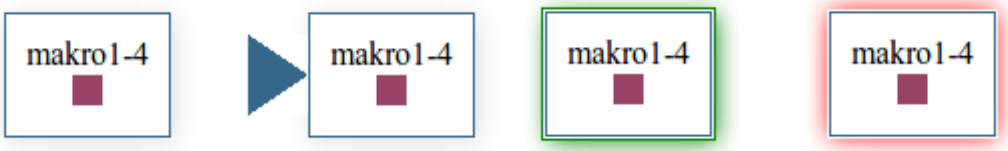
Pokud makro používá více symbolů, než je v páskové abecedě vytvářeného stroje, tak ho nelze přidat. Více symbolů z makra by se muselo mapovat na jeden symbol stroje, a to by mohlo vést k nedeterministickému Turingově stroji.

Po přidání makra do seznamu programů vlevo je ještě nutné přidat jeho kopii přetáhnutím myší do požadované pozice. Dvojklikem na něj nebo po kliknutí v seznamu programů ho lze editovat stejně jako hlavní program. Abecedy nemá svoje,

ale používá ty z hlavního programu.

Kliknutím na makro v seznamu programů ho lze přejmenovat, přitom se přejmenují i jeho instance, tak aby byly pořád ve tvaru 'jméno makra-číslo makra'. Dále je možné vyvolat editaci komentáře nebo uložit makro do paměti prohlížeče, použít ho pak lze při vytváření jiného stroje při dalším spuštění, nebo také hned v jiné záložce nebo okně stejného prohlížeče.

S jednotlivými instancemi makra v programu Turingova stroje lze zacházet stejně jako se stavy. Jediná výjimka je při přejmenovávání maker. Makro lze označit jako počáteční nebo koncové, případně může být i breakpointem. Grafické zobrazení je podobné stavům a je zobrazeno na obrázku 3.12.



Obr. 3.12: Makra jsou zvýrazněna stejně jako stavy, zleva doprava je to tedy obyčejné makro, počáteční makro, koncové makro a makro s breakpointem.

Znak

Aby bylo makro ještě více univerzální, lze použít při čtení z pásky a zápisu na ní znak #. Při čtení funguje jako $ELSE$, tedy pokud není splněna jiná podmínka, přejde se do dalšího stavu pomocí přechodu, ve kterém je v podmínce #. Pokud je tento symbol jako zapisovaný, tak stroj zapíše právě přečtený znak.

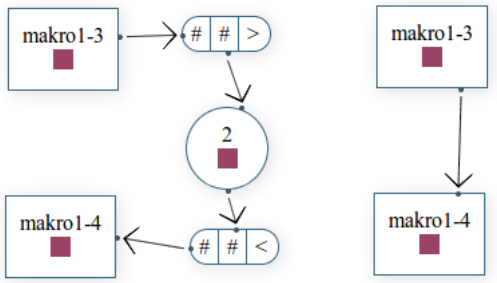
Při vhodném použití v makru pak bude toto makro správně pracovat, i když se jeho pásková abeceda rozšíří.

Přechod z makra do makra

Přechod z makra do makra, při kterém Turingův stroj nevykoná žádnou činnost, lze udělat například tak, jak je ukázáno na obrázku 3.13(a).

Elegantnějším způsobem je vytvoření standardního přechodu z makra do makra a jeho následné smazání klávesou DELETE. V případě, že na obou koncích je makro, smaže se podmínka a akce stroje, ale spojení mezi makry zůstane. K porovnání je tato možnost na obrázku 3.13(b).

Takto vytvořený přechod nelze označit myší a lze ho smazat dvojklikem.

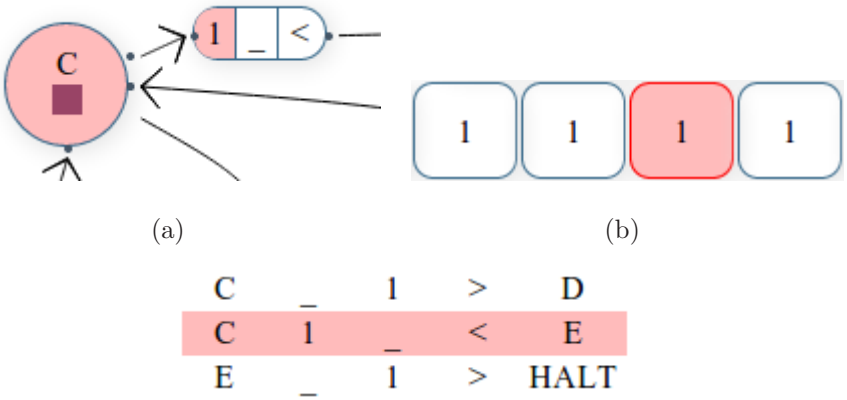


(a) (b)

Obr. 3.13: Přechod z makra do makra vytvořen pomocí dvou podobných způsobů. V případě (a) pouze stroj provede dva kroky navíc, které ale funkci stroje nezmění.

3.4 Zobrazení činnosti Turingova stroje

Při zapnutí simulace se na prvním symbolu ze vstupního slova na pásce se zobrazí hlava Turingova stroje a jako aktivní se zvýrazní počáteční stav nebo makro stroje. Pokud v dalším kroku dojde k přechodu do jiného stavu, je v tomto přechodu zvýrazněn aktuální symbol pod hlavou stroje. Ukázka zvýraznění jednotlivých prvků je na obrázku 3.14. Počítadlo provedených kroků se nastaví na nulu.



Obr. 3.14: Ukázka zobrazení aktuálního stavu stroje. Zvýrazněn je aktuální stav a právě splněná podmínka (a). Na pásce je zvýrazněna pozice hlavy (b) a v seznamu instrukcí je zvýrazněna následující prováděná instrukce.

Vpravo od přepínače pro simulaci je pět prvků pro její ovládání, posuvník pro rychlost ( ) a tlačítka pro reset ( ), spuštění ( ), krok ( ) a stop ( ). Tlačítko krok provede přechod z jednoho zobrazeného stavu do jiného, do provedených kroků se ale započítávají pouze opravdové kroky Turingova stroje (s podmínkou, zápisem a posunem hlavy) a ne například přechod z makra do jeho prvního stavu.

Tlačítko pro spuštění bude v intervalech podle zadané rychlosti automaticky provádět přesně to, co dělá tlačítko pro krok. K zastavení může dojít několika způsoby, stiskem tlačítka stop, dosáhnutím koncového stavu, breakpointu, nebo po přejití do stavu pomocí přechodu na kterém je breakpoint. Pokud pro daný stav a symbol není definován přechod, stroj se také zastaví, tentokrát ale vstupní slovo zamítne. Také se může stát, že stroj nikdy sám nezastaví.

Při resetování se znovu nastaví hlava na počáteční pozici, stejně jako aktuální stav na počáteční. Vstupní slovo na pásce se nastaví takové, jaké bylo při spuštění simulace.

V režimu simulace lze měnit program, je tedy možné ho psát postupně, ale nelze měnit symboly na pásce.

4. Implementace simulátoru

Jak je uvedeno v předcházejících kapitolách, ideální simulátor by měl být dostupný na webu bez instalace. Toto omezení vylučuje Flash, Javu a nějaké další možnosti. Zbývá tak jako skoro jediná možnost Javascript. Pro snadnější práci s Javascriptem jsem použil knihovnu jQuery.

V další části jsou detailněji popsány použité knihovny, použitá architektura aplikace, způsob uložení stroje v programu a popsány některé z použitých algoritmů.

4.1 Použité technologie a knihovny

4.1.1 Knihovny jQuery a jQuery UI

jQuery[1] je knihovna pro Javascript, která usnadňuje práci hlavně s vybíráním elementů v objektu DOMDocument a přidáváním událostí. Místo dlouhých názvů metod jako například: `document.getElementsByClassName(name)` lze použít s jQuery jednoduší zápis: `$(".name")`. Podobně lze třeba i při výběru podle `id` nebo typu elementu použít stejný zápis, jako v CSS.

Po výběru pak lze jednoduše pomocí metod `attr` a `html` nastavovat a získávat jednotlivé atributy a obsahy elementů.

Knihovna jQuery UI[2] je rozšíření jQuery o prvky uživatelského rozhraní a animované efekty.

Obě knihovny mají licenci MIT.

[1] `http://jquery.com/`

[2] `http://jqueryui.com/`

4.1.2 Knihovna jsPlumb

Knihovna jsPlumb[3] slouží k propojování prvků UI pomocí různých typů čar nebo šipek. Při každé změně polohy, nebo rozměru některého prvky se všechny spojení s ním přepočítají. Lze například nastavit, aby spojení vycházelo vždy se strany směřující k druhému spojovanému prvku. Také je možnost reagovat na události, pokud uživatel například klikne na spojení, nebo přes něj přejede myší. Pomocí knihovny lze nastavit některé elementy jako počáteční a některé jako koncové, uživateli je tím umožněno mezi těmito elementy myší vytvářet a přesunovat spojení, současně je možné na tyto události reagovat programově.

Pro vykreslovaní čar je možné zvolit SVG, Canvas nebo VML. Jako podpůrnou knihovnu je nutno mít jednu z následujících: jQuery, MooTools nebo YUI3. Při zvolení jQuery je pro podporu drag and drop potřeba i knihovna jQuery UI. Její licence jsou MIT a GLPLv2 a autorem je Simon Porritt.

4.1.3 Springy

Springy[4] je algoritmus napsaný v JavaScriptu pro automatické vytvoření rozložení grafu na 2D plochu. Pro výpočet souřadnic používá fyzikální simulaci pružin. I když umožňuje animace a přepočítávání souřadnic při každé změně, tak používám pouze finální vypočtené souřadnice. Kdyby se stavy Turingova stroje neustále přesunovaly, ztrácela by se přehlednost.

Použitá licence je MIT a autor je Dennis Hotson.

4.2 Popis použitých návrhových vzorů

Při návrhu objektově orientovaného programu lze často narazit na podobné struktury tříd, které řeší vždy stejný problém. Proto byly vytvořeny návrhové vzory, aby se usnadnila jejich identifikace už při návrhu a zrychlil se tím jeho proces. Nejsou

[3] `http://jsplumb.org/`

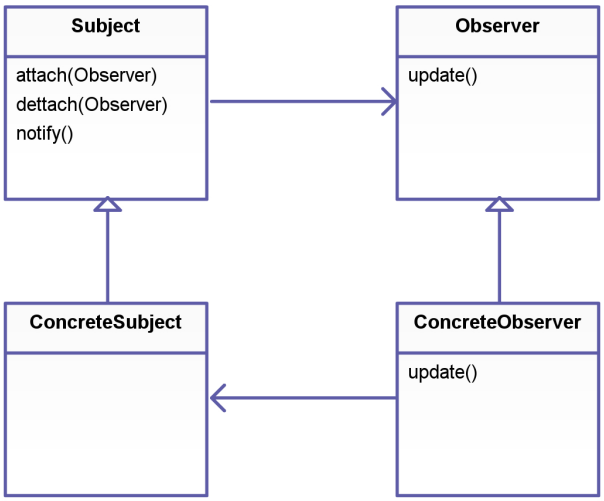
[4] `http://getspringy.com/`

to tedy knihovny pro jednotlivé jazyky, ale spíše návody pro propojení jednotlivých tříd.

Jsou uvedeny například v knize [2], kde je jich vysvětleno a popsáno celkem 23. Je to také jeden ze zdrojů, ze kterých jsem vycházel při jejich popisu.

V následující části jsou popsány návrhové vzory, které jsem v aplikaci použil, nebo které se k ní alespoň nějakým způsobem vztahují.

4.2.1 Návrhový vzor Observer



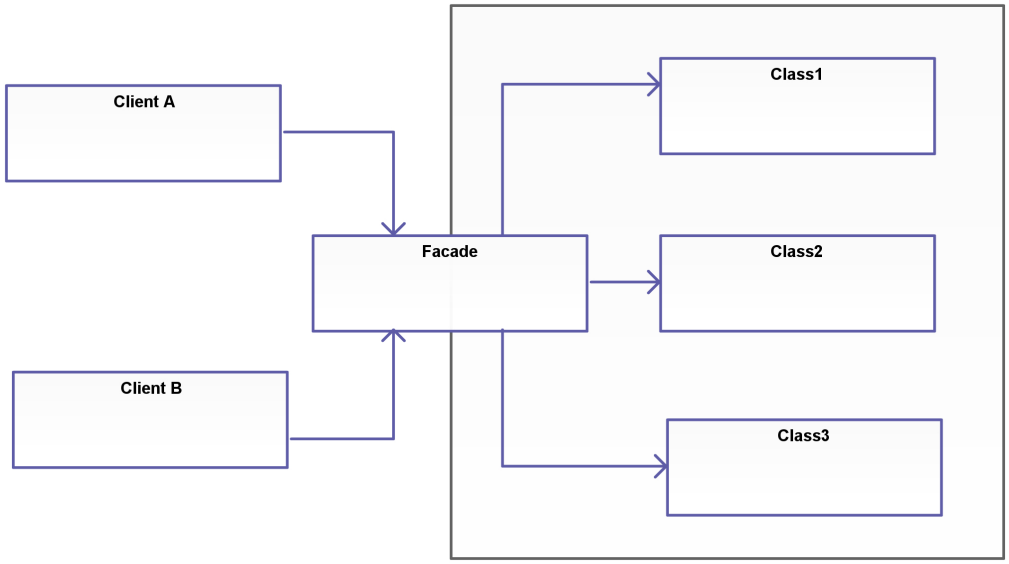
Obr. 4.1: Návrhový vzor Observer, třída ConcreteObserver sleduje změny ve třídě ConcreteSubject a reaguje na ně metodou update().

Vzor Oberver je použitelný v situaci, kdy je množina více objektů (observer), které by měly být automaticky upozorněny na právě vzniklou událost. Zároveň by ale objekt, ze kterého upozornění pochází (observable), neměl být závislý na této množině objektů.

Observable má metodu $attach(observer)$, kterou volá observer, který chce monitorovat jeho změny. Dále má observable metodu $notify$, kterou upozorní všechny jeho sledující observery na nějakou změnu.

Diagram tříd pro tento návrhový vzor je na obrázku 4.1.

4.2.2 Návrhový vzor Facade



Obr. 4.2: Návrhový vzor Facade, odděluje klienty od složitější struktury tříd.

Fasáda v architektuře označuje vnější stěnu stavby, tedy jak budovu vidí okolí. Podobně v programování je Facade návrhový vzor, který odděluje okolní třídy od tříd, které používá Facade. Vnitřní struktura Facade se tak může měnit, pokud zachová stejné metody. Příklad diagramu tříd s Facade je na obrázku 4.2.

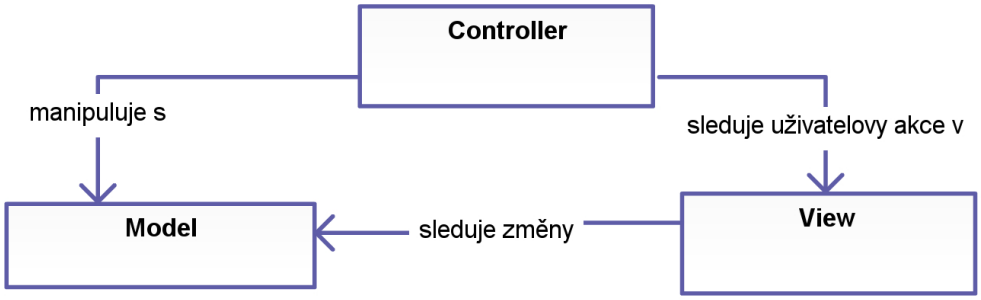
4.2.3 Architektura MVC

MVC je spíše než návrhový vzor konkrétní použití vzoru Observer a případně i Facade. Dobře je tato architektura popsána v článku [3], který ale vznikl v roce 1988 ještě před samotným uceleným popsáním těchto návrhových vzorů, takže v něm nejsou takto pojmenovány. Například následující věta zjednodušeně popisuje vzor Facade:

> „Models hold onto a collection of heir dependent objects.“

Podobně by zjednodušený popis vzoru Observer mohl znít následovně:

> „When a model has changed, a message is broadcast to notify all of its dependents about the change.“



Obr. 4.3: Diagram tříd pro architekturu MVC. Rozděluje aplikaci na model, view a controller.

Architektura MVC rozděluje aplikaci na tři typy částí: model, view a controller. V aplikaci je vždy každá použita minimálně jednou, ale lze použít například více view pro zobrazování více způsoby, nebo na více místech. Jejich charakteristiky jsou následující:

- Model reprezentuje informace, se kterými aplikace pracuje. Fungovat by měl samostatně i bez existence následujících částí. Jeho jeho složitost může sahat od jednoduchého datového typu až ke komplexnímu objektu čítajícímu velké množství tříd, kdy už se vyplatí využít vzor Facade.
- View slouží k zobrazování modelu. Dále model sleduje a patřičným způsobem reaguje na jeho změny. Sleduje i události od uživatele posílá je controlleru. Od modelu si může vyžádat jeho data, ale nesmí ho měnit.
- Controller reaguje na události od uživatele změnou modelu, případně view. Může měnit model, ale měl by to dělat pouze pomocí jeho metod, aby view pak mohlo zareagovat na tyto změny v modelu.

Průběhu jedné akce uživatele

U každého vstupu od uživatele aplikace provede několik kroků, končící zobrazením výsledku akce. Pak opět čeká na uživatelský vstup. Detailněji jsou tyto kroky popsány následně:

1. Uživatel klikne na nějaký grafický prvek.
2. Viev pošle událost dál controlleru.
3. Controller požádá model o provedení akce.
4. Controller může případně i změnit view, pokud to není část zobrazující model.
5. Model provede danou akci nějakou změnou svých dat.
6. View sleduje model a změny si všime.
7. View na změnu zareaguje změnou zobrazení.
8. Čekání na akci uživatele, po které se začne prvním krokem.

Existuje i možnost, že se model mění sám v závislosti na čase. Pak probíhá komunikace jen mezi modelem a view a uživatel nemusí aktivně zasahovat.

4.3 Použití návrhových vzorů v aplikaci

Při návrhu architektury aplikace jsem se inspiroval architekturou MVC, což je jedna z možností, jak vytvořit grafické uživatelské rozhraní. Model v tomto případě představuje Turingův stroj a jmenuje se `TS`. Protože Turingův stroj vzhledem k jeho složitosti netvoří jedna třída, použil jsem i návrhový vzor Facade. Implementaci Turingova stroje pak bude možno změnit a stačí zachovat jeho metody, případně i atributy. Například by šlo vytvořit verzi s minimem tříd a optimalizovanou pro větší rychlost simulace, bez nutnosti změn tříd view a controller.

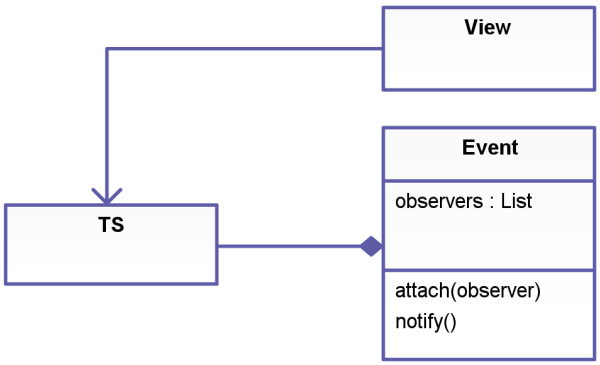
Objekty reprezentující model, view a controller se vytvoří v tomto pořadí po načtení stránky, tedy:

```
var ts = new TS();
var view = new View(ts);
var controller = new Controller(ts, view);
```

Model by měl fungovat bez view, ale i s více objekty view. Zároveň by ale každý objekt view měl reagovat na změny v modelu. Toto lze řešit například pomocí návrhového vzoru Observer.

V této aplikaci je to třída `Event`, zobrazena na obrázku 4.4, která má dvě metody: `attach` a `notify`. Při vytváření třídy `TS` se vytvoří i její instance pro jednotlivé typy

událostí, například pro přijetí slova je to `this.prijetiSlova = new Event();`. Pokud se pak Turingův stroj dostane do koncového stavu hlavního programu, tak se tato událost vyvolá: `this.prijetiSlova.notify({slovo:prijateSlovo});`, ale bez view se sama nijak neprojeví.



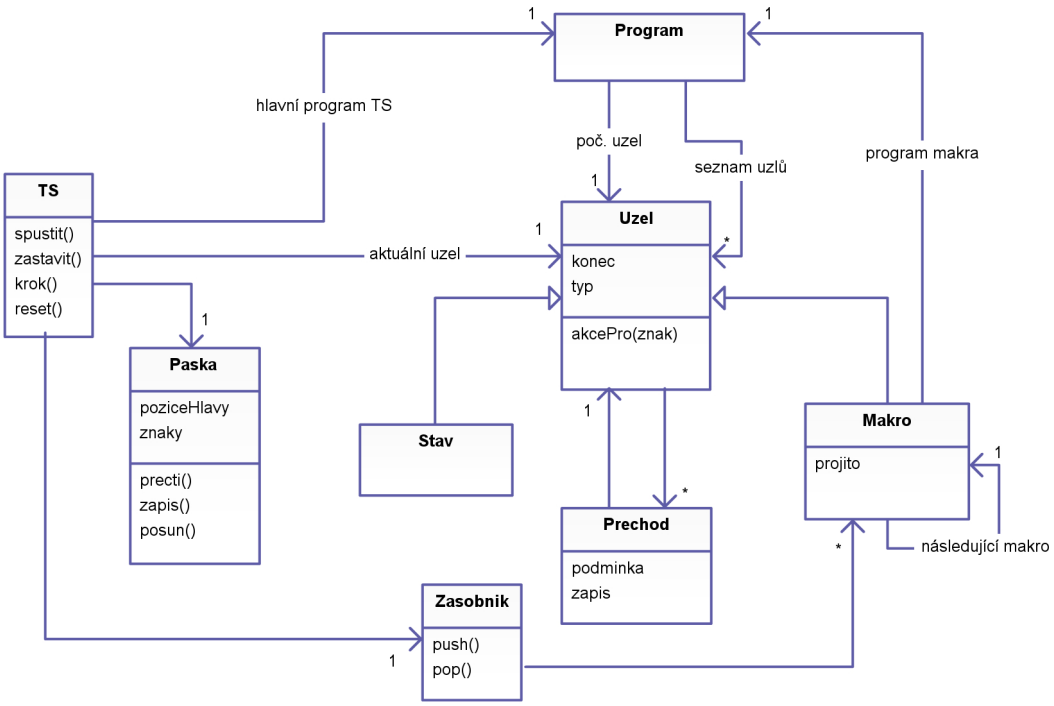
Obr. 4.4: Použití třídy event, která je implementací návrhového vzoru Observer v JavaScriptu. Místo přiřazení observeru, který by pak provedl update() je přiřazena přímo tato metoda.

Ve view je při jeho vytvoření kód, ve kterém je zapsána reakce na přijetí slova, což je zobrazení zprávy na obrazovku se slovem, které bylo přijato:

```
ts.prijetiSlova.attach(function(sender, data) {
     _this.showMessage("Přijetí slova '"+data.slovo+"'","green");
    })
```

(Identifikátor `_this` je pro referenci na view, `this` by odkazovalo na prováděnou funkci.)

Podobně se metoda `notify` volá při každé zobrazitelné změně modelu a view na ni vhodným způsobem reaguje. Podobným způsobem i view oznamuje události od uživatele controlleru.



Obr. 4.5: Zobrazeny třídy, které reprezentují Turingův stroj. Třída TS je použita jako facade a současně model v architektuře MVC.

4.4 Hlavní třídy v aplikaci

Diagram použité architektury MVC je na obrázku 4.3, pouze model mám pojmenovaný jako `TS`. Diagram pro třídu Turingova stroje a na něm závislých tříd je na obrázku 4.5. Následuje popis jednotlivých tříd, zmíněny a zobrazeny na obrázku jsou hlavně metody pro samotnou funkcionalitu Turingova stroje.

4.4.1 Třída Paska

Třída Paska slouží v uchování znaků na pásce a pozici hlavy na ní. Pro funkci stroje jí stačí metody `posun(smer)`, `precti()` a `zapis(znak)`. Další metody jsou pro editaci a uložení a načtení pásky po resetu stroje. Jednotlivé znaky pásky jsou uloženy jako prvky asociativního pole `znaky`. První znak vstupního slova má index `0`, indexy znaků vlevo jsou záporné a indexy znaků vpravo kladné. Pokud pro daný

index žádný znak není, vrátí se prázdný symbol, reprezentovaný znakem `_`.

4.4.2 Třídy Uzel, Stav a Makro

Třída pro uzel by měla být abstraktní třída reprezentující společné vlastnosti stavů a maker. JavaScript ovšem nemá abstraktní třídy a vlastně ani obyčejné třídy a dědičnost, pouze objekty. Proto používám pouze třídu Uzel. Makro lze od stavu odlišit jednoduše tím, že má nějaký program, zatímco stav ho nemá. Pro větší přehlednost má ale i každý stav atribut `typ`, která nabývá hodnot `stav`, nebo `makro`.

Dále má každý uzel atribut `konec`, označující koncový stav programu, a seznam přechodů vedoucích z tohoto uzlu. V případě, že jde o makro, může mít jako atribut i ihned následující prováděné makro.

V každém kroku Turingův stroj potřebuje získat prováděnou akci pro dvojici stav - znak, $\delta(q, X)$. Proto má třída uzel metodu `akcePro(znak)`, která vrací tři potřebné údaje: zapisovaný znak, směr posunu hlavy a nový stav (p, Y, D).

4.4.3 Třída Zásobník

Dvě makra můžou mít stejný program, ten je v aplikaci uložen pouze jednou. Při vstupu do programu makra se proto uloží aktuální stav (konkrétní instance makra) do zásobníku a po dokončení programu makra je tento stav obnoven.

4.4.4 Třída Přechod

V objektech třídy přechod jsou podmínky a instrukce (zápis a posun hlavy) pro vlastní činnost Turingova stroje. Z každého uzlu jich může vést libovolný počet a každý přechod vede pouze do jednoho stavu.

4.4.5 Třída Program

Třída Program reprezentuje hlavní program Turingova stroje (reference ze třídy TS), nebo také program jeho makra (reference ze třídy Uzel). Jako atributy má vlastně jen počáteční stav daného programu (`pocUzel`) a seznam uzlů programu

(`seznamUzlu`). Seznam uzlů je nutný pro uložení nedosažitelných uzlů (například ihned po jejich vytvoření).

4.4.6 Třída TS

Tato třída funguje jako model v architektuře MVC. Všechny třídy patřící k Turingově stroji, jsou tedy přístupné přes ní. Její nejdůležitější metody jsou `spustit`, `zastavit`, `krok` a `reset`, které ovládají chod Turingova stroje. Dále obsahuje metody pro přidávání a mazání jednotlivých stavů, přechodů, maker, a tak dále.

4.4.7 Uložení souřadnic

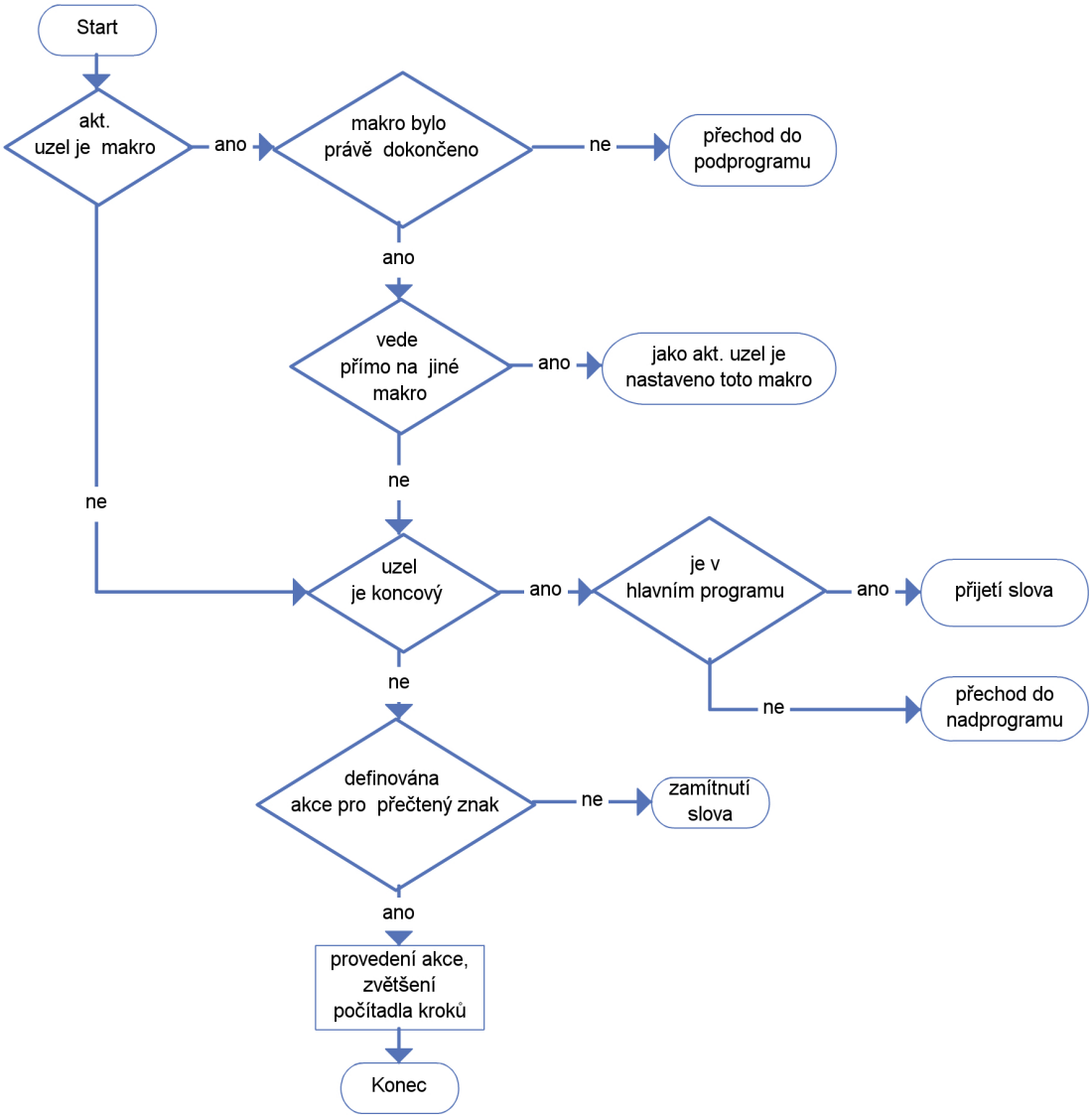
Pro grafické vykreslení stroje je také nutné znát souřadnice jednotlivých prvků. Turingův stroj by samozřejmě fungoval i bez nich, jen by nebylo moc dobře vidět, co vlastně dělá. Pro každý uzel a přechod jsem tedy přidal atributy `x` a `y` k jejich odpovídajícím třídám.

4.5 Metoda TS.krok()

Nejdůležitější metodou Turingova stroje je `krok()`, jejímž voláním se bude postupně vykonávat jeho program. Vždy provede maximálně jednou přečtení znaky z pásky, zapsání nového, posun hlavy o jedno políčko a přesun na nový uzel.

Vývojový diagram metody je na obrázku 4.6. Při spuštění stroje se metoda `krok()` volá v cyklických intervalech a zastaví se pouze přijetím, nebo zamítnutím slova, případně stroj může zastavit uživatel vnějším zásahem.

Při přechodu do jiného programu (podprogramu nebo nadprogramu) se mění atribut `TS.aktuaktualniPohled`, ve kterém se uchovává zobrazovaný program. Vždy je tedy na obrazovce zobrazena prováděná část celého programu.



Obr. 4.6: Vývojový diagram metody krok. Při spuštěném stroji se cyklicky provádí tato metoda, dokud nedojde k přijetí nebo zamítnutí slova.

4.6 Spuštění a zastavení stroje

Turingův stroj se spouští metodou `TS.spustit(rychlost)`, pro konstantní interval mezi kroky je použita metoda okna `window.setInterval(code,millisec)`. Následující ukázky kódu jsou v konstruktoru třídy TS:

```
this.spustit = function(rychlost) {
  this.bezi = true;
  this.running = setInterval(function(){
    _this.krok();
  },rychlost);
};
```

Pro zastavení Turingova stroje stačí interval `this.running` vymazat metodou `clearInterval(this.running)`. Rychlost je možné měnit i za běhu, řešeno je to pomocí zastavení a spuštění stroje s jinou rychlostí:

```
this.zmenitRychlost = function(rychlost) {
  if (this.bezi) {
    this.zastavit();
    this.spustit(rychlost);
  }
}
```

4.7 Převod Turingova stroje na binární tvar a zpět

Turingův stroj v binárním tvaru nemá makra a nemůže obsahovat znak `#`. Dále musí mít přesně jeden koncový stav. V následujícím textu je ukázán postup převedení stroje s makry na binární tvar.

Nejprve je nutno přejmenovat jednotlivé stavy tak, že se před ně přidá jméno makra, ve kterém se nalézají a znak `#` před každým jménem makra. Například `#toFirst-1#Start` značí stav pojmenovaný `Start` v makru `toFirst`.

Pomocí rekurzivního průchodu přes všechny stavy a makra se určí počáteční a koncové uzly pro celý program. Koncové stavy se všechny přejmenují na `#`, protože je není třeba odlišovat a musí existovat pouze jeden.

Poté je nutno vytvořit tabulky přechodů mezi stavy a přímých přechodů mezi makry. Dále už se pak pracuje jenom s těmito tabulkami. Pokud by tedy na kon-

cový stav nic nevedlo, vytvoří se přechod z nepojmenovaného stavu do stavu `#`. Na podmínce a akci nezáleží, protože tyto nově vytvořené stavy jsou nedosažitelné.

V následujícím kroku se zpracuje tabulka pro přímý přechod mezi makry. Bude v ní například `#makro-1#S -> #makro-2#E`. Z koncového stavu prvního makra se tedy ihned přejde do počátečního stavu druhého makra. Ve všech přechodech tedy půjde nahradit první údaj `#makro-1#S` údajem druhým `#makro-2#E` a činnost stroje se nezmění. Po nahrazení je možné tuto tabulku smazat.

V dalším kroku je nutné nahradit znaky `#` na pozici podmínky a zápisu. Pokud `#` zastupuje 1 znak, stačí ho pouze nahradit, pokud jich zastupuje více je potřeba tento přechod zkopírovat a v každém znak `#` nahradit jinou správnou variantou.

Tím vznikne tabulka přechodů bez maker bez znaků `#`. Následuje očíslování jednotlivých stavů a znaků v abecedách. Nejmenším číslem je označen počáteční stav a největším koncový stav. U znaků mívá číslo 1 prázdný symbol, který je v každé abecedě.

Pak už jen stačí projít jednotlivé přechody a přidávat správný počet nul a údaje a přechody oddělovat jedničkami.

Převod z binárního tvaru zpět je o dost jednodušší. Stačí pouze rozložit řetězec jedniček a nul znaky `111`, tím zůstane v prostřední části program. Na jednotlivé přechody se rozdělí s pomocí znaků `11` a podobně přechody na jednotlivé údaje pomocí znaku `1`. Znaky se dekódují pomocí seznamu znaků páskové abecedy a stavy se pojmenují svým číslem. Stav s nejmenším číslem bude počáteční a stav s největším koncový. Výsledkem je program Turingova stroje bez maker a symbolů `#`, který ale rozpoznává stejný jazyk jako před převodem na binární tvar a zpět.

4.8 Použití algoritmu Springy pro rozmístění uzlů

Při importu stroje z binárního tvaru nebo z nějakého jiného simulátoru nebudou v datech stroje souřadnice jednotlivých přechodů nebo stavů. Pro automatické rozmístění, které bude vypadat lépe, než náhodné je použit algoritmus v JavaScriptu pojmenovaný Springy.

Springy používá svůj formát tříd pro uložení grafu. Je tedy potřeba vytvořit novou proměnou graf a naplnit ji mými daty:

```
var graph = new Graph();
//pro každý uzel
  uzel.graph = graph.newNode();
  //pro každý přechod z tohoto uzlu
    prechod.graph = graph.newNode();
```

Pak je nutné přidat hrany mezi těmito uzly grafu:

```
//pro každou hranu
  graph.newEdge(uzel.graph,prechod.graph);
  graph.newEdge(prechod.graph,prechod.vedeNaStav.graph);
```

Tím je vytvořen graf ve formátu, kterému Springy rozumí a lze vytvořit a spustit fyzikální simulaci pro výpočet pozic jednotlivých uzlů. Springy podporuje i animaci tohoto výpočtu a s pomocí vlastní třídy `renderer` umožňuje jakékoliv její vykreslování.

Pro vytvoření simulace slouží třída `layout`, která má jako parametr graf a několik fyzikálních proměnných. S pomocí `layout(render,done)` pak lze spustit samotnou simulaci s použitím definované vykreslovací třídy (rendereru) a po skončení výpočtu se provede funkce `done`. Můj program překreslování v průběhu výpočtu nepoužívá pro zvýšení jeho rychlosti.

Po skončení výpočtu je u každého uzlu souřadnice x a y, které je nutno přetransformovat do souřadnic, které se používají u vykreslování programu Turingova stroje. Naleznou se nejvyšší a nejnižší souřadnice v obou osách a posunou se a vynásobí tak, aby graf byl vykreslen tak akorát do části pro graf.

4.9 Ukládaní a načítání strojů

Třída `TS` má metody pro export svých dat do textového řetězce a stejně tak i pro import. Tyto textové řetězce tvoří program Turingova stroje v textovém nebo

binárním tvaru, dle zvolené metody. Samotné ukládání do souboru nebo do aplikace provádí třída `controller`.

Nahrání souboru z disku počítače do aplikace lze pouze přes HTML element `input` typu `file`, jehož vzhled nelze snadno měnit a navíc ho každý prohlížeč zobrazuje jinak. V aplikaci je mu z těchto důvodů nastavena průhlednost na 100 procent a na stejné místo je vloženo tlačítko pro nahrání souboru.

Ukládání do souboru probíhá přes server a PHP pomocí pluginu pro jQuery, v tento moment je tedy potřeba být online. Při opětovném uložení stejného stroje se znovu objeví dialog pro uložení, pomocí JavaScriptu totiž nelze přímo ukládat na konkrétní místo v disku počítače nebo z něj i číst.

Pro rychlejší uložení a načtení lze stroje a makra ukládat do paměti prohlížeče nazvané LocalStorage. To funguje jako asociativní pole, lze tedy ukládat klíče a k nim hodnoty. V aplikaci jsou pro každý uložený program použity tři hodnoty: komentář k programu, pásková abeceda a program stroje v textovém tvaru.

I když je komentář a abeceda součástí programu, tak jsou odděleny. Je pak rychlejší jejich načtení do seznamu při přidávání nového makra, protože není potřeba parsovat textový tvar programu.

4.9.1 Kontrola správnosti souborů

Při zadávání Turingova stroje v textovém nebo binárním tvaru může uživatel snadno udělat chybu v syntaxi. Proto jsou soubory kontrolovány pomocí regulárních výrazů.

Pro textový tvar se vstupní řetězec rozdělí na jednotlivé řádky a pak se každý kontroluje zvlášť, takže při chybě se vypíšou pouze chybné řádky.

Pokud řádek začíná znakem `#` nebo `;`, tak jde o začátek nového programu, nebo komentář a tyto řádky moc kontrolovat nelze.

Na dalších řádcích pak může být přechod bez souřadnic, přechod se souřadnicemi, název stavu stav se souřadnicemi nebo přechod z makra do makra. Každý řádek je pak porovnán s následujícími výrazy. Pokud se neshoduje ani s jedním, tak je označen za chybný.

```
^[^ ]* [^ ]* [^ ]* [^ ]* [^ ]*$
^[^ ]* [^ ]* [^ ]* [^ ]* [^ ]* [0-9]* [0-9]*$
^[^ ]* [0-9]* [0-9]*$
^[^- ]*-[0-9]* [^- ]*-[0-9]$
```

V binárním tvaru tvoří program stroje řetězec jedniček a nul, není ale libovolný. Zkontroluje se najednou pomocí následujícího výrazu:

```
^11(1[0]*1[0]*1[0]*1[0]*1[0]*1)*11$
```

4.10 Grafická reprezentace stroje

Zobrazování uzlů v grafické podobě stroje je realizováno pomocí kombinace HTML a CSS. Spojovací šipky mezi jednotlivými prvky jsou vykreslovány pomocí SVG.

U každého uzlu je kromě jeho jména uloženo i identifikační číslo (id), které vzniká s jeho vytvořením a toto číslo nelze změnit. Uživatel ho nevidí, ale je použito pro identifikování jednotlivých uzlů v DOMDocumentu a následné manipulaci s nimi.

Pomocí tříd CSS jsou pak zobrazovány jednotlivé typy uzlů (stav nebo makro) a jejich vlastnosti (počáteční, koncový nebo breakpoint). Kód HTML pro počáteční stav stroje může vypadat například následovně:

```
<div class="state ui-draggable ui-droppable aktivni"
id="state-0" style="left: 230px; top: 107px;">
  <div class="name">b</div>
  <div class="ep" id="jsPlumb_1_99"></div>
  <div class="pocUzel" style="visibility: visible;"></div>
 </div>
```

Třídy začínající `ui` a `jsPlumb` si přidají samy odpovídající knihovny. Element `div` pro počáteční uzel je přítomen v každém uzlu, ovšem jen u toho jednoho správného je mu nastavena viditelnost. Samotná šipka ukazující počáteční uzel je vlastně kus rámečku.

Kód pro přechod mezi stavy je velmi podobný kódu pro uzel:

```
<div class="trans ui-draggable" id="trans-0"
style="left: 451px; top: 208.5px;">
  <div class="cond">_</div>
  <div class="write">0</div>
  <div class="move">&gt;</div>
  <div class="ep" id="jsPlumb_1_100"></div>
</div>
```

Políčko pásky tvoří pouze jeden `div`:

```
<div id="policko-0" class="policko ">_</div>
```

4.11 Použití jsPlumb pro spojování uzlů

Knihovna jsPlumb je používána ke spojování jednotlivých prvků programu stroje pomocí šipek, které se překreslují dle pozice spojovaných prvků.

Nejprve je nutno provést inicializaci, ve které je možné nastavit vykreslovací režim a výchozí hodnoty. Pro vykreslování lze použít SVG, HTML5 Canvas nebo VML. Pokud se režim nenastaví, je použito vykreslování pomocí SVG.

```
jsPlumb.setRenderMode(jsPlumb.SVG);
```

Podobně lze nastavit i ostatní režimy. Dále je nastavení jednotlivých vykreslovacích prvků.

```
jsPlumb.importDefaults({
  Connector:[ "StateMachine", { curviness:20 } ],
  PaintStyle:{ strokeStyle:"#000000", lineWidth:1 },
  Endpoint:[ "Dot", { radius: 2} ],
  Anchors:[
    [ "Continuous", {}],
```

```
    [ "Continuous", {}]
  ],
  ConnectionOverlays : [
    [ "Arrow", {
      location:1,
      id:"arrow",
      length:10,
      foldback:0.1
  } ] ], });
```

Connector je v tomto případě typ čáry (spojení) spojující jednotlivé elementy, paintStyle určuje její vzhled. Endpoint je koncový bod spojení. Anchor je místo, kam lze připojit spojení, v tomto případě je použit typ `Continuous`. Ten spojení připojí na jednu ze stran v závislosti na pozici elementu na druhém konci spojení. Zapsán je v kódu dvakrát, jednou pro začátek spojení a podruhé pro konec spojení. Jako poslední je ConnectionOverlays, pomocí tohoto parametru lze nastavit dodatečné vykreslovací prvky, v tomto případě se jedná o šipku.

Poté se nastaví počáteční elementy, ze kterých bude možné myší vytvořit spojení.

```
$(".ep").each(initEP);


initEP = function(i,e) {
  var p = $(e).parent();
  jsPlumb.makeSource($(e), {
    parent:p,
});};
```

Předchozím kódem se jako počáteční nastaví elementy třídy `ep`, které jsou v každém stavu a makru. Parametr `parent` složí k označení elementu, ve kterém začne samotné spojení, v tomto případě to bude makro nebo stav.

Podobně se nastaví i koncové elementy, na kterých lze myší spojení zakončit.

```
jsPlumb.makeTarget(jsPlumb.getSelector(".makro"), {
  dropOptions:{ hoverClass:"dragHover" },
});
```

Podobně jako makro se nastaví i stav jako koncový.

Pro možnost posunu myší je možné použít funkci jsPlumb, v tomto případě pro přechod:

```
jsPlumb.draggable(jsPlumb.getSelector(".trans"));
```

Při přesunu jednoho elementu se ale hýbe pouze přesunovaný a při požadavku na přesun více elementů současně je nutné použít funkci z jiné knihovny a ručně vyvolávat funkce na překreslování spojení.

Při přesunu stavů se o polovinu přesouvají i k němu připojené přechody. Použil jsem funkci jQuery UI nazvanou `draggable`. Ta má jako jeden z parametrů událost `drag`, která se vyvolá při posunu myši. Jako reakce na tuto událost je volána funkce `jsPlumb.repaint(element)`, která překreslí všechna spojení k požadovanému elementu.

Spojování jednotlivých elementů poté lze provádět pomocí myši přetažením z elementu třídy `ep` do elementu s třídou `makro` nebo `state`. Takto se vytvoří šipka ze stavu do stavu nebo makra. Na tuto událost je navázána metoda `jsPlumbConnection`. V ní se vytvoří element `trans`, ve kterém se nastaví podmínka a akce Turingova stroje a spojí se se správnými stavy a makry. Dále se v ní aktualizuje model, který ale neoznámí žádnou změnu, protože vizuální změny už provedla knihovna jsPlumb.

Vytvářet spojení je JavaScriptem metodou `jsPlumb.connect(attrs)`. Použití je například následující:

```
jsPlumb.connect({source:"element1", target:"element2"})
```

Zdrojový je element s atributem id rovnému `element1` a koncový s id `element2`. Dalšími atributy lze měnit typ čáry, barvu, způsob připojení atd.

Podrobnější popis všech metod a funkcí je v dokumentaci ke knihovně jsPlumb [4].

4.12 Vstupy od uživatele

Správně by v architektuře MVC mělo docházet k aktualizaci view až po změně modelu. Při použití jsPlumb a vytvoření spojení myší ale dojde nejprve k vykreslení spojení a poté k aktualizaci modelu. Tato aktualizace by už ale neměla způsobit další aktualizaci ve view.

U použití formulářových elementů HTML dochází k podobné situaci. Například `input` typu `text` se aktualizuje okamžitě po stisku klávesy. Navíc událost `onchange` se spouští pouze při přesunu kurzoru na jiný prvek.

Pro současnou změnu názvu stavu v textovém vstupu na graficky zobrazeném stavu proto jsou použité metody třídy `document` pro detekci stisklých klávesy.

Stejné metody jsou použity i u zápisu na pásku a na přechody, ve kterých dochází k aktualizaci view teprve až po změně modelu. Ve view je kód, který detekuje stisklé klávesy a oznamuje je spolu s prvkem, který byl označen:

```
$(document).keypress(function(event) {
  znak = String.fromCharCode(event.charCode);
  if ($("#paska .aktivni").length != 0) {
    pozice = Number($("#paska .aktivni").attr("id").substring(8));
    _this.keyPressed.notify({prvek:"paska", znak:znak,
                             pozice:pozice});
    return;
  };
  //následují další podmínky pro jiné aktivní prvky
};
```

Controller tyto změny sleduje a předává modelu informace o tom, co má udělat:

```
this._view.keyPressed.attach(function (sender,data) {
  switch (data.prvek) {
    case 'paska':
      if (_this._ts.zapisZnakNaPozici(znak,pozice))
        _this._view.nastavAktivniPolicko(pozice+1);
      break;
    //další case
}
```

V metodě `ts.zapisZnakNaPozici(znak,pozice)` se kontroluje zda znak je v páskové abecedě, pokud ano, tak se na pásku na požadovanou pozici zapíše a upozorní na změny:

```
TS.prototype.zapisZnakNaPozici = function(znak,pozice) {
  if (this.inVstupniAbeceda(znak)) {
    this.paska.znaky[pozice] = znak;
    this.paska.zapsaniZnaku.notify({znak : znak, pozice : pozice});
    return true;
  } else {
    this.setMessage.notify({
      text:"Znak '"+znak+"' není ve vstupní abecedě",
      color:"red"})
    return false; }};
```

Teprve ve view se pak zobrazená páska zaktualizuje:

```
ts.paska.zapsaniZnaku.attach(function (sender,data) {
  $("#paska #policko-"+data.pozice).text(data.znak);
});
```

Podobným způsobem jsou provedeny všechny ostatní změny modelu.

5. Závěr

V rešeršní části práce jsem srovnal několik nalezených simulátorů Turingova stroje. Dvě nejlépe hodnocené dosáhly 78 bodů. Aplikaci jsem psal tak, aby co nejlépe splňovala požadavky na ideální simulátor a získala tak více bodů než 78.

Simulaci lze provádět po krocích ručně nebo automaticky (25). Aplikaci lze spustit z prohlížeče bez instalace jakýchkoliv doplňků (20). Turingův stroj je možné nakreslit v grafické podobě (20). Je možné uložit Turingův stroj do souboru a načíst ho z něj (15). Pro zjednodušení návrhu lze použít makra (5), několik základních je už v aplikaci po jejím prvním načtení. Na úvodní stránce aplikace je na výběr z 5 příkladů (5). I bez hodnocení složitosti ovládání můj simulátor Turingova stroje získá celkem 95 bodů.

Aplikace je dostupná na serveru `kaja.nti.tul.cz`, případně pomocí přímé adresy `kaja.nti.tul.cz/~ozogan/turing/`. Spolu s dalšími podobnými aplikacemi na tomto serveru je určena pro studenty a usnadnění výuky předmětů zabývajícími se automaty a formálními jazyky.

Literatura

[1] HOPCROFT, John E. *Introduction to automata theory, languages, and computation.* 3rd ed. Boston: Pearson - Addison-Wesley, 2007, xvii, 535 s. ISBN 03-214-5536-3.

[2] SHALLOWAY, Alan a James TROTT. *Design patterns explained: a new perspective on object-oriented design.* Boston, Ma.: Addison-Wesley, 2002, xxvii, 334 p. ISBN 02-017-1594-5.

[3] KRASNER, Glen E. a Stephen T. POPE. A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of object-oriented programming.* 1988, roč. 1, č. 3. ISSN 0896-8438. Dostupné z: `www.ics.uci.edu/~redmiles/ics227-SQ04/papers/KrasnerPope88.pdf`

[4] PORRITT, Simon. *JsPlumb 1.3.16 - documentation* [online]. 2012 [cit. 2013-05-04]. Dostupné z: `http://jsplumbtoolkit.com/doc/usage.html`

A. Obsah přiloženého CD

Struktura a obsah adresářů je následující:

/doc Text bakalářské práce ve formátu pdf.

/src Zdrojové kódy aplikace ve stejné struktuře, jako jsou na serveru.

/src/css CSS soubory definující vzhled aplikace.

/src/ico Ikony použité v aplikaci.

/src/js Soubory controller.js, TS.js, view.js a init.js obsahují kód odpovídajících částí aplikace.

/src/js/lib Obsahuje podadresáře s jednotlivými použitými knihovnami (jQuery, jsPLumb a Springy).

/src/js/lib/HTML5Slider Obsahuje soubor html5slider.js, který zobrazí posuvník pro ovládání rychlosti simulace ve Firefoxu stejně, jako v ostatních prohlížečích.

/src/machines Stroje použité jako příklady na úvodní stránce.

/src/macros Makra, která se načtou do paměti při spuštění aplikace.